

Fork Entropy: Assessing the Diversity of Open Source Software Projects' Forks

Liang Wang
State Key Laboratory for
Novel Software Technology,
Nanjing University, Nanjing, China
wl@nju.edu.cn

Zhiwen Zheng
State Key Laboratory for
Novel Software Technology,
Nanjing University, Nanjing, China
zwZheng@smail.nju.edu.cn

Xiangchen Wu
State Key Laboratory for
Novel Software Technology,
Nanjing University, Nanjing, China
xchenwuhhu@gmail.com

Baihui Sang
State Key Laboratory for
Novel Software Technology,
Nanjing University, Nanjing, China
DZ21330024@smail.nju.edu.cn

Jierui Zhang
State Key Laboratory for
Novel Software Technology,
Nanjing University, Nanjing, China
jieruizhang@smail.nju.edu.cn

Xianping Tao
State Key Laboratory for
Novel Software Technology,
Nanjing University, Nanjing, China
txp@nju.edu.cn

Abstract—On open source software (OSS) platforms such as GitHub, forking and accepting pull-requests is an important approach for OSS projects to receive contributions, especially from external contributors who cannot directly commit into the source repositories. Having a large number of forks is often considered as an indicator of a project being popular. While extensive studies have been conducted to understand the reasons of forking, communications between forks, features and impacts of forks, there are few quantitative measures that can provide a simple yet informative way to gain insights about an OSS project's forks besides their count. Inspired by studies on biodiversity and OSS team diversity, in this paper, we propose an approach to measure the diversity of an OSS project's forks (i.e., its fork population). We devise a novel fork entropy metric based on Rao's quadratic entropy to measure such diversity according to the forks' modifications to project files. With properties including symmetry, continuity, and monotonicity, the proposed fork entropy metric is effective in quantifying the diversity of a project's fork population. To further examine the usefulness of the proposed metric, we conduct empirical studies with data retrieved from fifty projects on GitHub. We observe significant correlations between a project's fork entropy and different outcome variables including the project's external productivity measured by the number of external contributors' commits, acceptance rate of external contributors' pull-requests, and the number of reported bugs. We also observe significant interactions between fork entropy and other factors such as the number of forks. The results suggest that fork entropy effectively enriches our understanding of OSS projects' forks beyond the simple number of forks, and can potentially support further research and applications.

Index Terms—Open Source Software, Diversity of Forks, Fork Entropy, Rao's Quadratic Entropy, Mining Software Repositories

I. INTRODUCTION

Forks play a central role in modern pull-based Open Source Software (OSS) development as precursors of making contributions to source repositories through pull-requests [1]–[4]. On social coding platforms such as GitHub and GitLab, forks are created for various purposes including but not limited to

archiving, learning, fixing bugs, adding new features [1], [5]. Existing studies suggest that forks create increased opportunities for community engagement and voluntary participation [6], [7]. Having a large number of forks is often considered as an indicator of an OSS project being popular [7], [8]. However, merely having a large number of forks does not necessarily imply a productive and healthy OSS project. Inefficient fork practices can lead to lost, rejected, or redundant contributions as pointed out in existing studies [3], [6]. Understanding the factors related to an OSS project's fork efficiency can provide valuable insights to help project maintainers and contributors to diagnose and improve their fork practices [6].

To gain insight into and improve forking efficiency, prior research has explored the relationship between a project's fork efficiency and its own characteristics, such as modularity and coordination [6]. In contrast, this work takes a different perspective by examining the *population of forks* created from a project. Specifically, we propose to measure the *diversity of an OSS project's forks* by evaluating their differences in terms of modifications made to project files with the Rao's quadratic entropy—a measure widely adopted in quantifying population and ecological diversity [9], [10]. Our interest in understanding the diversity, beyond the simple count, of forks is inspired by the importance of diversity in different related domains: 1) *Biodiversity* is shown to have significant effects on species' productivity, resilience, competition, and survival [11]. In this work, we follow the promising approach of applying concepts and measures from ecological studies to the field of OSS research [12], [13], and study how a project's fork population diversity is linked to its development. 2) *OSS Teams Diversity*, including social diversity [14], gender and tenure diversity [15], [16], culture and country diversity [17], [18], and linguistic diversity [19], has been extensively studied and shown to be significantly linked to team performance. While the above work focuses on the diversity of OSS contributors, our study examines the diversity of the artifacts they create,

i.e., the forks. 3) Previous work in *redundant change detection* [20] and *fork feature extraction* [21] has demonstrated how valuable information, such as patch content, changed file lists, clusters of features, etc, can be derived from forks' changes to project files. In line with this prior work, we focus on the diversity of forks regarding their modifications to project files. To summarize, this study broadens the scope of diversity discussed in OSS development to the artifacts, i.e., forks, by introducing a new quantity called **fork entropy** to understand a project's fork population beyond simply the number of forks. To the best of our knowledge, this work is among the first to quantify the diversity of an OSS project's fork population, and study its correlation with pull-based OSS development.

While anyone can fork from a public OSS repository, in this study, we specifically focus on forks and contributions from external contributors who cannot directly commit to OSS projects mainly for the following three reasons. First, with the OSS model, anyone can potentially be an external contributor, resulting in a large population in this group [22]. Second, external contributors' contribution through pull-requests and bug reports are critical during the development and maintenance of OSS projects [4], [8], [23]. Investigating external contributors' forks and contributions is crucial for understanding their effects on OSS projects' productivity and fork efficiency. Third, fork and pull-request is the primary, if not the only, approach for external contributors to contribute since they do not have write access to the projects [22], [24], making our study more focused on fork-related features and performance. It is worth noting that while we restrict our investigation to external contributors' forks, we may also include forks owned by core members if the project strictly follows the pull-based development model, i.e., fork and contribute back to the source repository through pull-requests instead of directly pushing commits into the source repository, for all contributors including the core developers with write access. We solely examine external contributors' forks in this study to maintain a focused study, and aim at answering the following research questions using data retrieved from fifty OSS projects' on GitHub.

RQ1: *What is the correlation between an OSS project's fork entropy and its external productivity?*

In this work, we operationalize an OSS project's external productivity [8], [15], [25], [26] as the number of commits integrated from external contributor-owned forks into the project's source repository. The results of regression analysis suggest a project's fork entropy is significantly and positively correlated to its external productivity, especially for projects younger in ages.

RQ2: *How does an OSS project's fork entropy relate to its acceptance rate of external pull-requests?*

The acceptance rate of external pull-requests serves as a useful proxy for measuring fork efficiency [6] and can provide insight into a project's openness to external contributions¹. In this study, we operationalize the acceptance rate as the

proportion of closed pull-requests raised by external contributors from their respective forks that were ultimately merged into the project. Our analysis indicates that fork entropy has a significant, positive correlation with the acceptance rate of external pull-requests, particularly when those pull-requests involve modifications to frequently changed 'hot' files.

RQ3: *What is the correlation between fork entropy and OSS projects' number of reported bugs?*

The number of reported bugs is commonly used as a proxy for evaluating software quality in projects [8], [26]–[29], and in this study, it is operationalized as the monthly reported bugs. Our analysis reveals a significant, negative correlation between fork entropy and the number of reported bugs in OSS projects. We also find that a high level of fork entropy is linked to a reduced rising trend of bug-reporting issues with the increasing number of forks in OSS projects.

Through the above studies, we conclude that the diversity of OSS projects' fork population plays an important role in understanding the development of OSS projects under the pull-based model. And the proposed fork entropy metric is an effective and useful indicator in assessing fork diversity, which shows significant correlations to key productivity and quality indicators about OSS development and maintenance. In summary, this work makes the following contributions.

- We propose to measure the diversity of OSS projects' fork populations to gain insights about pull-based OSS development, which provides a novel point of view about the forks and diversity about OSS projects.
- We devise the fork entropy metric based on Rao's quadratic entropy, and show properties of the metric including symmetry, continuity, and monotonicity, which makes the metric effective in measuring fork diversity.
- We conduct empirical studies to reveal the correlations between fork entropy and the external productivity, the acceptance rate of external pull-requests, and number of reported bugs in OSS projects, which demonstrate the usefulness in understanding the diversity of forks.
- We discuss the implications of fork entropy in understanding and guiding practices of pull-based social coding for OSS development.

The rest of this paper is organized as follows. Sec. II introduces the background and related work. Sec. III presents the process of calculating the proposed fork entropy metric and shows its properties. In Sec. IV, we present the design of empirical studies, and report the results in Sec. V. We discuss the implications of this work, and the threats to validity in Sec. VI. Finally, we conclude the paper in Sec. VII.

II. BACKGROUND & RELATED WORK

This section introduces the background and related work.

A. Studies on Forks and Pull-Based OSS Development

The pull-based development model is a modern paradigm for software development that is particularly well-suited to geographically distributed teams [4]. This workflow can be broken down into seven steps, which include: forking, cloning,

¹<https://chaoss.community/kb/metric-change-request-acceptance-ratio/>

editing, syncing, pushing, submitting, and evaluating [2], [3]. By providing a code base and a set of tools for task management, code review, and DevOps, the pull-based model simplifies participation and lowers entry barriers for external contributors compared to traditional patch-based models [4].

Forks can be created for many different reasons or intentions. In [1], Jiang et al. summarizes developers' reasons and preferences of creating forks, and OSS projects' attracting characteristics of getting forked. They discover various reasons of forking including contributing back with pull-requests, fixing bugs, adding new features, keeping copies, etc. Stănciulescu et al. conduct a case study using the Marlin project to explore the reasons, benefits, are challenges of forks [30]. According to whether they will contribute back to the source repository, forks can be classified as "hard" forks [31], [32], independently developed forks (IDFs) [33], or divergent forks [34] that are not intended to be merged back, and "social" forks [32] that are created to make contributions.

The many forks created from a source repository (with different intentions above) form the fork population studied in this work. The concept is closely related to the concept of software family in existing work [31], [34], [35]. In [35], Brisson et al. study the communication in a software family with respect to following, pull-requests, and issues, and analyze the correlation between such communication and a project's star counts. In [34], Businge et al. explore the characteristics including the size, package dependencies, categories, etc., of software families, as well as their code propagation practices. Software families composed of hard forks are studied in the above work [34], [35]. In [31], Hadian et al. study the evolution and communication between repositories in a family of projects (forks). They discover differences between hard and social forks in terms of activity, deviation of dependencies, and communication. In this work, we study the diversity of a projects' contributors-owned fork population created on GitHub. Although we do not discriminate hard and social forks in our studies, we are more interested in contributing forks [33] by analyzing the correlation between the forks' diversity and contributions received in the source repository.

For owners of contributing forks that follow the pull-based model, they can fork and develop in their own forked repositories before contributing back to the source repository through pull-requests. However, coordination issues such as misalignment and conflict among developers can lead to inefficiencies in forking practices, which include lost contributions, rejected pull requests, redundant development, and fragmented communities as summarized by Zhou et al in [6]. Consequently, it is essential to understand the factors, and develop tools to aid efficient forking practices. In [6], Zhou et al. explore characters of the source repository such as modularity, and coordination mechanisms on the efficiency of forking practices. To better understand the forks of OSS projects, tools are developed to visualize source code changes in forks [36], and identify features from commits in a projects forks [21]. In this work, we propose a new metric called fork entropy to facilitate studies of the diversity of a forks based

on their modification to files, which offers a new perspective to understand an OSS project's fork population.

B. Studies on External Contributors and Pull-Requests

In this work, we focus on forks owned, and contributions made by external contributors because we focus on the diversity and contributions related to OSS projects' forks. Unlike core developers who have write access, and can directly commit into a source repository, external contributors make contributions to projects through patches or pull-requests [4], [22]. In [22], Padhye et al. categorize code committers into core, external, and mutant, and find that the number of external committers of projects developed by popular scripting languages is comparable with the number of core committers. After inspecting the pull-requests opened in 2013 in GitHub, Gousios et al. conclude that pull-requests is a way of projects to get external contributions because 73.07% of the pull-requests have been merged using facilities provided by GitHub [4]. In [8], Vasilescu et al. study the influence of adopting continuous integration (CI) on the of acceptance of core and external (non-core) developers' pull-requests, as well as the impact on software quality. They discover an improvement in projects' ability in integrating external contributions without sacrificing code quality after adopting CI. By investigating the Eclipse community, Sinha et al. discover factors, such as demonstration of knowledge and skill in bug repositories, can influence the promotion of external developers to core committers [37]. In brief, contributions made by external contributors through pull-requests play an important role in OSS development and maintenance which worth studying. Moreover, external contributors are a suitable group for our study on fork diversity and its correlation on OSS contributions because, unlike core developers with write access, they need to fork a repository before making contributions.

C. Studies on Diversity Related to OSS Projects

Most existing software engineering studies related to our work focus primarily on the diversity of team members in OSS projects. Gender diversity has been widely examined and found to have positive effects on project growth [14], team productivity [15], and community health [16]. Studies have also reported a positive correlation between the country diversity of team members and project growth [14], and the impact of tenure diversity on team productivity has been suggested [15]. Additionally, Daniel et al. measured the reputation and role diversity of participants and found positive effects on community engagement and market success [17]. However, as far as we know, there is currently a dearth of formal measures of fork diversity. Moreover, the diversity metrics presented above are typically measured using either the Blau index [38] or the coefficient of variation [39]. While the Blau index is well-suited for measuring diversity in categorical variables (such as gender [15] and country [14]), and the coefficient of variation can measure dispersion in numerical variables (like tenure and reputation), neither method is ideal for calculating the proposed metric of fork entropy due to

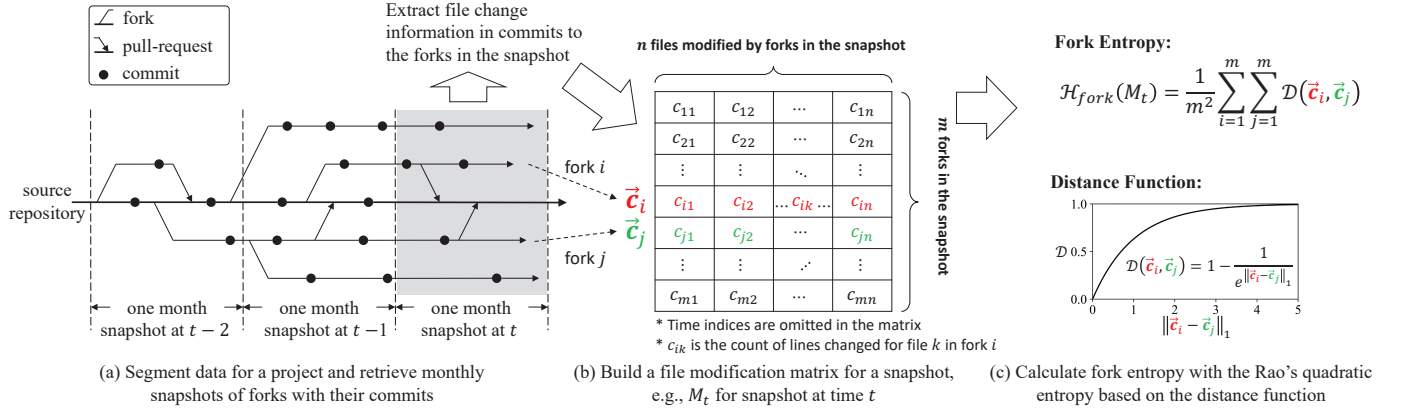


Fig. 1. Overall process of calculating fork entropy.

limitations on the forms of data associated with understanding the fork populations of OSS projects.

Diversity is widely acknowledged as a multifaceted concept, which can be categorized into three main components: richness, evenness, and disparity [40]. Richness refers to the absolute number of species in a population, evenness describes the distribution of species abundances, and disparity denotes the differences among species [41]. Due to its complexity, diversity has been operationalized using different metrics, depending on the specific application. For instance, while species richness is a commonly used indicator of diversity that emphasizes the richness component, the Gini coefficient, which measures income inequality, highlights the disparity component [41]. In the context of OSS development, forks may naturally differ due to differences in expertise, experience, and intentions among developers [42]. Therefore, we focus on the disparity component of diversity and quantify the differences in file modifications among forks. To this end, we employ Rao's quadratic entropy, which measures the expectation of dissimilarity between two samples randomly taken from a population [9]. This approach has been widely used in studies of population and ecological diversity [9], [10]. Further details on how we calculate fork entropy can be found in Section III.

III. FORK ENTROPY

This section presents the proposed measure of fork diversity with fork entropy.

A. Overview

Fig. 1 illustrates the overall process of calculating fork entropy to measure the diversity of forks created for an OSS project. As shown in Fig. 1(a), for each project involved in our study, we first collect data about its forks, and construct a series of snapshots of *fork populations* with predefined time intervals. We select projects from GitHub for easy data collection. We empirically set the time interval to one month in this work after consulting the literature [26]. Second, as shown in Fig. 1(b), we build a *file modification matrix* to contain the changes made by the forks in snapshot at time

t , denoted as M_t . To keep the notations concise, we slightly abuse the notations to omit time index t when referring to the file modification matrix for a snapshot in the rest of the paper. Finally, as shown in Fig. 1(c), we calculate the fork entropy with the Rao's quadratic entropy defined on the pairwise *distance function* that quantifies the difference between each pair of forks given a file modification matrix. Detailed steps are as follows.

B. Construct Fork Populations in Snapshots

The first step is to construct the fork population as shown in Fig. 1(a). Given an OSS project, we first locate its source repository, e.g., `tensorflow/tensorflow`, in the database. We then perform a breadth first search using the 'forked_from' key in the database to retrieve the direct forks and indirect forks (i.e., 'forks of forks') of the source repository in an iterative manner until all forks of the project are retrieved. The retrieved forks are segmented by time intervals with a fixed duration (empirically set to a month in this work) to get a series of snapshots. Each snapshot contain forks with their commits received during the time interval. For a fork to be included in a snapshot, it must: 1) have at least one commit with file modifications during the snapshot's time interval; and 2) is owned by an external contributor who does not have write access to, or have privileges to close issues or pull-requests in the source repository [6]. We determine a contributor to be external if he / she has never directly commit into a repository, or performed any privileged actions such as closing issues opened by other users. A fork can be included in multiple snapshots if it meets the above criteria in different time intervals, but with different commits. We restrict our scope to forks owned by external members of the project to conduct a focused study as introduced in the beginning of the paper. It should be noted that fork ownership is not a part of the proposed fork entropy metric. Following the above approach, the set of forks in each snapshot forms a *fork population* of the project during the corresponding time interval.

C. Build Fork File Modification Matrix for Each Snapshot

Next, we build a *file modification matrix* to contain the changes made by the fork population in a snapshot. In the matrix, a fork, e.g., the i -th fork, is encoded as a row vector:

$$\vec{c}_i = \langle c_{i1}, c_{i2}, \dots, c_{ij}, \dots, c_{in} \rangle^\top,$$

where n is the number of files in the project that are modified by one of the forks in the population, and $c_{ij} \in \mathbb{R}$ is the count of lines in file j which are modified by fork i during the time interval of the snapshot. Intuitively, \vec{c}_i is the fingerprint of the i -th fork in terms of modifications to project files. Let m be the number of forks in the population, we can obtain the *file modification matrix*, $M \in \mathbb{R}^{m \times n}$, by stacking the row vectors of all of the m forks as shown in Fig. 1(b). Because the set of files modified by fork populations in different snapshots are likely to be different, it is common that the number of columns, n , varies with different time intervals. We include only files that are modified by the fork population in a snapshot to guarantee that the file modification matrix does not contain rows or columns that are all zeros.

D. Calculate Fork Entropy with Rao's Quadratic Entropy

For each snapshot, we use the Rao's quadratic entropy with a distance function defined on the file modification matrix M to calculate the average degree of difference between forks in the population following Eq. (1).

$$\text{QE}(M) = \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m \mathcal{D}(\vec{c}_i, \vec{c}_j), \quad (1)$$

where $\text{QE}(M)$ denotes the Rao's quadratic entropy that estimates the expectation of difference between two individuals randomly selected from the population [9], M is the file modification matrix correspond to the snapshot (with the time index t omitted), m is the number of forks in the population, \vec{c}_i, \vec{c}_j are the i - and j -th row in M , respectively, and \mathcal{D} is a distance function that quantifies the degree of difference between two vectors as defined in Eq. (2) and visualized in Fig. 1(c).

$$\mathcal{D}(\vec{c}_i, \vec{c}_j) = 1 - \exp(-\gamma \|\vec{c}_i - \vec{c}_j\|_1), \quad (2)$$

where $\exp(-\gamma \|\vec{c}_i - \vec{c}_j\|_1)$ is the Laplacian kernel [43], $\|\cdot\|_1$ is the 1-norm, and γ is the hyperparameter used to adjust the sensitivity of the function to differences. We adopt the Laplacian kernel because it is a non-linear transform sensitive to slight change and performs excellently in many detection tasks, e.g., character recognition [44].

Practical considerations also led us to adopt the Laplacian kernel in our study. Our analysis revealed that many forks in our dataset contain only minor changes to project files—some forks modifying only a single line in a single file, similar to findings from previous research [45], [46]. Consequently, we observed a substantial number of small differences, resulting in a fork entropy distribution that roughly followed a bell curve. After testing various distance functions, including the Laplacian and Gaussian kernels [43], we ultimately selected

the distance function presented in Equation (2) based on its real-world performance.

By substituting Eq. (2) into Eq. (1), we have the Eq. (3) for fork entropy.

$$\mathcal{H}_{fork}(M) = \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m \left(1 - \exp(-\gamma \|\vec{c}_i - \vec{c}_j\|_1)\right), \quad (3)$$

where γ is set to 1 to compute the raw difference between two vectors in practice. With the above definition, it is trivial to see that $0 \leq \mathcal{H}_{fork} < 1$, and \mathcal{H}_{fork} takes the minimum value when the numbers of changed lines for all files modified by different forks are identical.

E. Properties of the Proposed Fork Entropy

Considering the basic axioms [41] of a diversity index and expectations in the particular context jointly, the following properties are met by the proposed fork entropy:

Symmetry. *The fork entropy is not related to the order of forks during its calculation.* It is straightforward to see \mathcal{H}_{fork} satisfies the symmetry property because its distance function \mathcal{D} is symmetric, i.e., $\mathcal{D}(\vec{c}_i, \vec{c}_j) = \mathcal{D}(\vec{c}_j, \vec{c}_i)$.

Continuity. *The fork entropy is a continuous function.* It is also easy to see that \mathcal{H}_{fork} is in a continuous interval $\mathcal{H}_{fork} \in [0, 1)$ by its definition.

Symmetry and continuity are two fundamental properties of a diversity index [41]. Next, we introduce monotonicity as a property to meet our goal of measuring fork diversity.

Monotonicity. *Adding a redundant (or distinctive) fork will decrease (or increase) fork entropy.* We first explain the terms before proving monotonicity. Given m existing forks and a new fork \vec{c}_{m+1} , Eq. (4) calculates the difference of the new fork to existing ones.

$$\tilde{\mathcal{D}}(\vec{c}_{m+1}) = \frac{1}{m} \sum_{i=1}^m \mathcal{D}(\vec{c}_i, \vec{c}_{m+1}). \quad (4)$$

We say \vec{c}_{m+1} is *redundant* if its difference to existing forks is less than the average difference among the existing m forks; in contrast, \vec{c}_{m+1} is *distinctive* if the new fork's difference to existing forks exceeds the average difference among the existing m forks.

Assuming that the vector for a new fork \vec{c}_{m+1} is added to an existing file modification matrix M that contains m forks to obtain a new matrix M' , we derive the new fork entropy of M' in Eq. (5).

$$\begin{aligned} \mathcal{H}_{fork}(M') &= \frac{1}{(m+1)^2} \sum_{i=1}^{m+1} \sum_{j=1}^{m+1} \mathcal{D}(\vec{c}_i, \vec{c}_j) \\ &= \frac{1}{(m+1)^2} \left(\sum_{i=1}^m \sum_{j=1}^m \mathcal{D}(\vec{c}_i, \vec{c}_j) + 2 \sum_{i=1}^m \mathcal{D}(\vec{c}_i, \vec{c}_{m+1}) \right) \\ &= \frac{m^2}{(m+1)^2} \mathcal{H}_{fork}(M) + \frac{2}{(m+1)^2} \sum_{i=1}^m \mathcal{D}(\vec{c}_i, \vec{c}_{m+1}). \end{aligned} \quad (5)$$

We denote $\mathcal{H}_{fork}(M') - \mathcal{H}_{fork}(M)$ as Δ and obtain Eq. (6) by substituting Δ into Eq. (5).

$$\Delta = \frac{2m+1}{(m+1)^2} \left(\frac{1}{m+0.5} \sum_{i=1}^m \mathcal{D}(\vec{c}_i, \vec{c}_{m+1}) - \mathcal{H}_{fork}(M) \right) \quad (6)$$

$$\approx \frac{2m+1}{(m+1)^2} \left(\tilde{\mathcal{D}}(\vec{c}_{m+1}) - \mathcal{H}_{fork}(M) \right).$$

According to Eq. (6), Δ is negative when \vec{c}_{m+1} is redundant and is positive when \vec{c}_{m+1} is distinctive. It means that fork entropy decreases after adding a redundant fork while increases after adding a distinctive fork. Consequently, fork entropy possesses the above properties and is valid to quantify the diversity of forks.

IV. METHODS FOR EMPIRICAL STUDIES

This section presents the variables, dataset, and analysis methods for the studies about the proposed fork entropy with respect to our research questions.

A. Variables

We measure a project's outcomes, including external productivity, the acceptance rate of external pull-requests, and number of reported bugs. We also introduce control variables relevant to those outcomes.

Outcome: external productivity. The number of commits is a widely used indicator of the productivity of OSS projects [8], [15], [25], [26]. In this work, we focus on external productivity that quantifies external contributor's contributions to a project, which is measured by the number of commits integrated into the project's source repository through pull-requests from external contributor-owned forks. The monthly external productivity is obtained to perform regression analysis with fork entropy in each snapshot to answer **RQ1**. A possible threat lies in that maintainers may change the origins of commits, for example, by "cherry-picking" in pull-requests [4], which can cause us to miss some contributions made by external contributors. Fortunately, we find such cases are rare in our dataset after manual inspections.

It is important note that software developers' productivity is a multifaceted concept which covers the activity, performance, efficiency, satisfaction and well-being, etc, of the developers as suggested in [47], and can be influenced by many factors beyond the ones studied in this paper such as the team sizes [48]. In this paper, we adopt the number of commits as a commonly used measure of productivity with respect to developers' activity, which should not be considered as a comprehensive measure of external developers' productivity.

Outcome: external pull-request acceptance rate. Researchers regard the acceptance rate of pull-requests as a crucial indicator of the development efficiency of OSS projects [6] because maintainers reject pull-requests that are obsolete, conflicting, duplicated, etc [4], [49], [50]. We focus on the acceptance rate of external pull-requests delivered from external contributor-owned forks to an OSS project's source repository, measured as the proportion of the merged pull-requests among closed ones. The list of merged and closed pull-requests can be

obtained from GitHub's rest API. The monthly acceptance rate of external pull-requests is calculated to perform regression analysis with fork entropy in each snapshot to answer **RQ2**. As many developers integrate pull-requests via other mechanisms rather than GitHub interface, the status of pull-requests is not very reliable reported by GitHub [4], [6]. We follow the heuristics first proposed by Gousios et al. [4] and subsequently refined by Zhou et al. [6] to determine pull-requests' status. A pull-request is considered been merged if any of the following conditions is met: 1) Maintainers perform a 'merged' action for the pull-request on GitHub. 2) The pull-request is closed by a commit using certain phrase conventions (e.g., `fixes #1234`) advocated by GitHub², or any of the last three comments of the pull-request refers to a commit indicating the merge of the pull-request³, and the commit exists in the source repository's commit history.

Outcome: number of reported bugs. The number of bugs per unit time is a popular proxy of code quality [8], [26]–[29]. We refer to [8], [26] to count emerging bug-report issues in each snapshot. We do not distinguish bug reports from core members or external contributors. To identify bug-report issues, we process issue titles and labels by lowercasing and Porter stemming [51] then search bug-related keywords, including *defect*, *error*, *bug*, *issue*, *mistake*, *incorrect*, *fault*, and *flaw*. If the title or any label of an issue contains at least one keyword, we mark it as a bug-report issue. The monthly code quality is assessed to perform regression analysis with fork entropy in each snapshot to answer **RQ3**. Since the count-based assessment of code quality relies heavily on the issue base, a threat arises if projects rarely utilize GitHub's default issue-trackers. Thus, we examine the number of issues in each project and exclude projects with few issues.

Control variables. Based on prior software engineering literature [4], [6], [8], [52] and our experience, we include the following factors potentially relevant to the above project outcomes as control variables.

- **NumForks** and **NumFiles**: The two variables are the number of forks and that of modified files, respectively. They jointly describe the shape of a file modification matrix. The more forks a project has, the more pull-requests are submitted by non-core developers [8].
- **ProjectAge**: The age of a project's source repository in days. The older the project, the fewer external pull-requests maintainers merge or reject [8].
- **NumStars**: The number of stars a project's source repository receives. This variable usually refers to the popularity of OSS projects. External contributors are more likely to contribute to more popular projects [8].
- **RatioOldContributors**: The ratio of external contributors with prior experience in successfully submitting pull-requests to a project's source repository. Core developers

²<https://github.blog/2011-04-09-issues-2-0-the-next-generation/>

³Comment matches regular expression `(merg|apply|appl|pull|push|integrat|land|cherry (-|s+)pick|squash) (ing|i?ed)`.

prefer to trust contributors they have worked with before [4], [6], [52].

- **RatioPRsWithTests:** The ratio of pull-requests that contain test cases. A pull-request has test cases if any file pathname contains ‘test’ [52]. Pull-requests that contain test cases are more likely to be merged [52].
- **RatioPRsTouchHotFiles:** The ratio of pull-requests that touch hot files. A hot file is that one modified by any merged pull-request in the past three months [4]. Existing studies suggest pull-requests that modify hot files are more likely to be accepted [4].

B. Data Collection

We collect data through GHTorrent [53] and GitHub REST API⁴, with the aid of the OSS Compass community [54]. We start by selecting the most popular five thousand projects from the May 2019 GHTorrent dump according to the number of stars a source repository receives. Then, we filter projects based on the following criteria.

- *Projects that do not develop software applications or frameworks are removed.* We remove projects that serve for document storage or course teaching. We examine project names and ‘README’ files by searching keywords, including *awesome*, *homework*, *assignment*, *course*, *note*, and *document*. If any keyword is found, we remove the project after manually rechecking. We also delete projects with no programming-language-specific files by looking at the file extensions.
- *Projects whose number of active forks or external pull-requests are less than one hundred are removed.* To ensure a project has sufficient forks and contributions, we retain projects that 1) contain at least one hundred active forks, i.e., forks that have pushed at least one commit into the forked repository [34], and 2) contain at least one hundred external pull-requests submitted by external contributors.
- *Projects whose issues are less than one hundred are also removed.* To reduce the threat to **RQ3**, we conservatively exclude projects with less than one hundred issues to ensure the remaining projects actively use the default issue-trackers on GitHub.

Among the 2533 candidate projects selected with the above criteria, we sample fifty projects⁵ to cover different application domains, which include ten application software (e.g., Atom/atom), two system software (e.g., kubernetes/kubernetes), sixteen web libraries and frameworks (e.g., angular/angular.js), nine non-web libraries and frameworks (e.g., tensorflow/tensorflow), and thirteen software tools (e.g., Microsoft/vscode).

⁴<https://docs.github.com/en/rest>

⁵A complete list of the fifty projects studied in this paper can be found at <https://github.com/wangliang-cs/fork-entropy-ase-2023-repos>.

C. Regression Analysis

We calculate monthly variables for each project from its creation to May 2019. All variables in each project snapshot compose an independent unit for regression analyses. We omit units with empty fork populations because fork entropy is meaningless without forks, even if it has a value of zero. Before fitting models with the data, we perform log-transform on control variables under skewed distributions to stabilize variance and reduce heteroscedasticity [55], including *NumForks*, *NumFiles*, *NumStars*, and *RatioOldContributors*. Then, we standardize fork entropy and all control variables to make the mean of each one is zero and the standard deviation is one, which makes all estimated coefficients of the model are on the same scale. Additionally, we manually examine distributions of outcome variables and conservatively remove about 1% of values as outliers to ensure the models are robust against outliers [56].

We build generalized linear mixed models (GLMMs) to analyze the correlations between fork entropy and the outcome variables. GLMMs inherit from generalized linear models (GLMs) to allow response variables from non-normal distributions and extend GLMs to include both fixed and random effects [57]. After exploratory data analysis, GLMMs are appropriate because the response variables in our data are non-normal and show apparent variability among projects. Specifically, fork entropy, control variables, and interactions between fork entropy and each control variable are modeled as fixed effects. To capture the project-to-project variability in the response (e.g., some projects naturally attract more external contributors and receive more contributions than others), we add a random-effects term for projects into the models. We also allow for deviations in the slope of the number of a project’s forks from the population values (i.e., we accept the possibility that, for example, projects with higher initial external productivity may, on average, be less strongly affected by the increase in fork counts). We use the `glmer` function provided by the `lme4` package [58] in R to build models. Following prior practices [6], [59], the Poisson and logistic regressions are specified for *count* (i.e., external productivity and number of reported bugs) and *ratio* (i.e., the acceptance rate of external pull-requests) response variables, respectively. We also explicitly set the denominator (i.e., the number of closed pull-requests) from the ratio as the `weights` parameter when modeling the acceptance rate of external pull-requests.

We check the collinearity among independent variables using the variance inflation factors (VIF below five is recommended [60]). All values are below 2 in our models, which means collinearity is not a problem in our data. We adopt the marginal R-squared (R_m^2) and the conditional R-squared (R_c^2) to assess the goodness-of-fit of the models. R_m^2 and R_c^2 describe the proportion of variance explained by the fixed effects alone and explained by the fixed and random effects together, respectively [61], [62]. In addition, we report the estimated effect, standard error, and significance level (i.e., *p*-value) for each model variable. We also report the results

TABLE I
FIXED EFFECTS OF EXTERNAL PRODUCTIVITY MODEL.

	Estimate (Std. Errors)	Chisq
(Intercept)	3.360 (0.136)***	
\mathcal{H}_{fork}	0.325 (0.005)***	5038.09***
NumForks	0.614 (0.075)***	67.84***
NumFiles	0.510 (0.006)***	7700.90***
ProjectAge	-0.176 (0.005)***	1097.94***
NumStars	0.032 (0.004)***	98.00***
RatioOldContributors	0.079 (0.003)***	600.72***
$\mathcal{H}_{fork}:\text{NumForks}$	-0.116 (0.006)***	383.30***
$\mathcal{H}_{fork}:\text{NumFiles}$	0.064 (0.005)***	147.55***
$\mathcal{H}_{fork}:\text{ProjectAge}$	-0.123 (0.004)***	1070.85***
$\mathcal{H}_{fork}:\text{NumStars}$	0.087 (0.005)***	311.58***
$\mathcal{H}_{fork}:\text{RatioOldContributors}$	0.015 (0.003)***	21.59***
AIC=65465.38; BIC=65558.12; $R_m^2=0.40$; $R_c^2=0.98$		
Num. obs.=3579; Num. groups: ProjectID=50		
*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$		
Log-transformed and standardized variables following Sec. IV-C.		

of ANOVA type-II analysis, and Akaike’s information criteria (AIC) [63] and Bayesian’s information criteria (BIC) [64] for each model using the `car` and `performance` package [65] in R, respectively.

V. RESULTS OF THE STUDIES

This section shows the results of regression analyses for our research questions.

A. RQ1: What is the correlation between an OSS project’s fork entropy and its external productivity?

To answer **RQ1**, we model the number of commits integrated from external contributor-owned forks into a project’s source repository as a function of fork entropy. In the regression, we control factors including the number of forks, the number of modified files, project age, the number of stars, and the ratio of external contributors with prior experience. Table I summarizes the regression results. We can see that the model is effective by explaining about 98% of the variance with the fixed and random effects together, which exceeds the fixed effects alone by approximately 58 percentage points.

The results in Table I suggest that fork entropy significantly and positively correlates to the external productivity of OSS projects (p -value below 0.001), with the third largest estimated coefficient just lower than the coefficients of NumForks and NumFiles. This result suggest that projects with more diverse fork populations integrate more commits created by external contributors. In addition, the controlled factors in the model are also significantly linked to external productivity. Table I shows that all controls except project age are positively correlated to the external productivity of OSS projects. As a result, OSS projects with more forks and modified files, higher popularity, younger in age, and more proportion of external contributors with prior experience generally correspond to higher external productivity.

We next analyze the interactions between fork entropy and the control variables in the model. As shown in Table I, all interactions show significance. The interactions between fork entropy and the number of forks and between fork entropy and

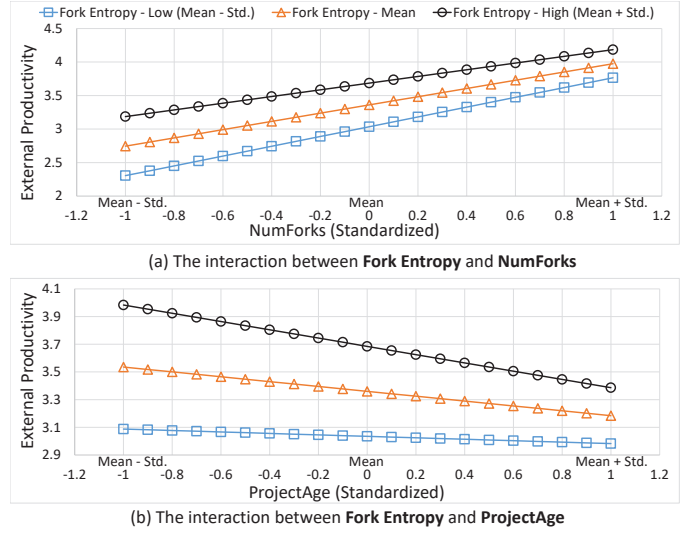


Fig. 2. Interactions in the external productivity model shown in Table I.

project age negatively correlate to the external productivity of OSS projects, and the remaining interactions show positive correlation with the response. We can also observe from the table that the two interactions negatively correlated to the response have the two largest effects as discussed below.

Fig. 2(a) illustrates the trends of external productivity with the increasing number of forks at low, middle, and high levels of fork entropy, respectively. We define the low, middle, and high levels of fork entropy respectively to correspond to its values with mean minus one standard deviation (Mean - Std.), mean, and mean plus one standard deviation (Mean + Std.) after consulting the literature [66]. In general, external productivity raises with the increase in NumForks. And the interaction suggests a higher level of fork entropy is associated with a lower growth rate of external productivity with respect to NumForks. With a fixed number of forks, an increase in a project’s fork entropy is related to an increased external productivity. This result suggests that, a project with a larger and more diverse population of forks have a higher probability of receiving contributions from the external contributors.

Fig. 2(b) depicts the interaction between fork entropy and project age. We can see that the external productivity generally decreases with the increasing of project age. The ratio of decreasing in external productivity with increased project age is amplified with higher levels of fork entropy. A possible explanation to this result is, younger projects under active development are possibly more open to accepting diverse contributions from external contributors than older projects. And projects older in age may have possibly entered a stage of stable maintenance, or slowly dying for lacking the capacity of handling diverse external contributions.

Fork entropy shows a positive and significant interaction between each of the other control variables including NumFiles, NumStars, and RatioOldContributors, respectively, as reported in Table I. We omit the details due to page limits.

In summary, we answer **RQ1** as follows. *Fork entropy*

TABLE II

FIXED EFFECTS OF EXTERNAL PULL-REQUEST ACCEPTANCE RATE MODEL.

	Estimate (Std. Errors)	Chisq
(Intercept)	-0.729 (0.195)***	
\mathcal{H}_{fork}	0.179 (0.015)***	145.07***
NumForks	-0.196 (0.158)	1.55
NumFiles	0.049 (0.020)*	11.54***
ProjectAge	0.490 (0.011)***	2710.74***
RatioOldContributors	0.614 (0.010)***	4578.80***
RatioPRsWithTests	0.135 (0.017)***	63.97***
RatioPRsTouchHotFiles	-0.091 (0.015)***	43.30***
$\mathcal{H}_{fork}:\text{NumFiles}$	-0.122 (0.012)***	97.07***
$\mathcal{H}_{fork}:\text{ProjectAge}$	-0.053 (0.009)***	38.38***
$\mathcal{H}_{fork}:\text{RatioOldContributors}$	-0.028 (0.009)**	10.01**
$\mathcal{H}_{fork}:\text{RatioPRsWithTests}$	-0.230 (0.011)***	421.33***
$\mathcal{H}_{fork}:\text{RatioPRsTouchHotFiles}$	0.140 (0.011)***	155.37***

AIC=46173.12; BIC=46272.04; $R_m^2=0.09$; $R_c^2=0.53$

Num. obs.=3579; Num. groups: ProjectID=50

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

Log-transformed and standardized variables following Sec. IV-C.

shows a significant, positive correlation with the external productivity of OSS projects. Further analysis uncovers significant interactions between fork entropy and other variables, including the number of forks, project age, number of files, number of stars, and ratio of old contributors. We observe that increasing fork entropy is related to a higher level of external productivity, particularly with respect to the number of forks. Additionally, our results indicate that for younger projects, increasing fork entropy shows a stronger positive relation with external productivity compared to older projects.

B. RQ2: How does an OSS project's fork entropy relate to its acceptance rate of external pull-requests?

To answer **RQ2**, we study the correlation between fork entropy and the acceptance rate of pull-requests submitted by external contributors. Here we calculate fork entropy based on a variant of the file modification matrix that only includes modifications involved in pull-requests. We perform the adjustment because pull-requests allow us to figure out which modifications are submitted to source repositories in each time interval. We exclude modifications made in forks but not included in pull-requests because they are largely invisible to maintainers who make decisions. We apply logistic regression for the acceptance rate by considering fork entropy. We control other factors, including NumForks, NumFiles, ProjectAge, RatioOldContributors, RatioPRsWithTests, and RatioPRsTouchHotFiles. Interactions between fork entropy and each control variable are also involved in the model. We ignore the interaction between fork entropy and NumForks because the control variable does not show a significant correlation with the response. Table II summarizes the model's results. The model explains about 53% of variability by including the fixed and random effects together, achieving a significant improvement compared to the fixed effects alone.

From Table II, we find fork entropy significantly and positively correlates to the acceptance rate of external pull-requests, despite that fork entropy has a smaller estimated effect of 0.179 compared with the estimated coefficients of Ra-

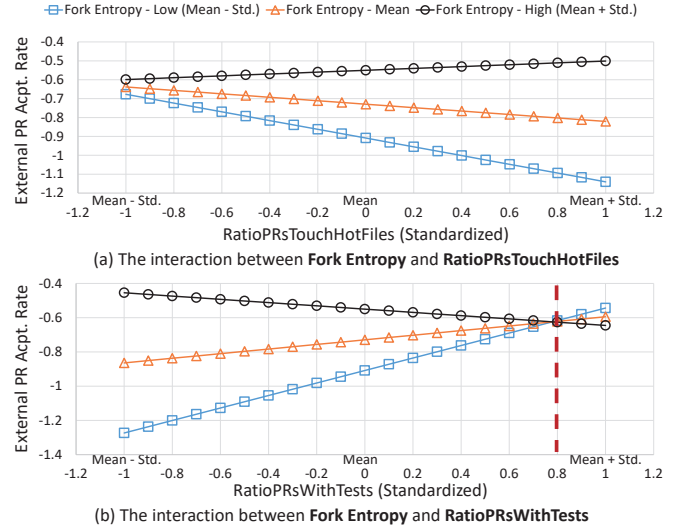


Fig. 3. Interactions in the acceptance rate model shown in Table II. Negative response values caused by the logit function.

tioOldContributors, ProjectAge, and NumForks of 0.614, 0.49, -0.196, respectively. However, the effect of NumForks fails to show statistical significance. Except for RatioPRsTouchHotFiles, the other four control variables are positively linked to the acceptance rate of external pull-requests.

For the model's interaction terms, only the interaction between fork entropy and the proportion of pull-requests that touch hot files show significant and positive correlation with the acceptance rate of external pull-requests. The remaining interactions are significantly and negatively correlated to the response. Furthermore, we analyze the interactions with relatively notable effects, i.e., the interaction between fork entropy and RatioPRsTouchHotFile, as well as its interaction with RatioPRsWithTests, as follows.

First, as illustrated in Fig. 3(a), the correlation between the proportion of pull-requests that touch hot files (RatioPRsTouchHotFiles) and external pull-requests' acceptance rate differs with different levels of fork entropy. The acceptance rate declines with RatioPRsTouchHotFiles when fork entropy is at a median or low level, and increases when fork entropy is high. A possible explanation is that, when fork entropy is at the median or low level, pull-requests are touching the same or similar sets of hot files, which result in conflicts and duplication among them, leading to a reduced acceptance rate. On the contrary, when fork entropy is at a high level, the pull-requests are touching different, less overlapped, hot files in the recent history. As a result, an increase in RatioPRsTouchHotFiles also leads to an increase in acceptance rate.

Next, Fig. 3(b) illustrates the interaction between fork entropy and RatioPRsWithTests on the acceptance rate of external pull-requests. We observe that an increase in the proportion of pull-requests that contain test cases is correlated to a higher acceptance rate when fork entropy is at a low level, which is consistent with existing observations that including test cases are helpful to make a pull request being accepted

TABLE III
FIXED EFFECTS OF NUMBER OF REPORTED BUGS MODEL.

	Estimate (Std. Errors)	Chisq
(Intercept)	2.632 (0.204)***	
\mathcal{H}_{fork}	-0.086 (0.006)***	222.49***
NumForks	0.356 (0.066)***	32.51***
NumFiles	-0.065 (0.007)***	82.53***
ProjectAge	0.078 (0.007)***	118.94***
NumStars	0.115 (0.006)***	332.64***
$\mathcal{H}_{fork}:\text{NumForks}$	-0.063 (0.008)***	62.36***
$\mathcal{H}_{fork}:\text{NumFiles}$	0.045 (0.007)***	40.92***
$\mathcal{H}_{fork}:\text{ProjectAge}$	0.032 (0.005)***	37.03***
$\mathcal{H}_{fork}:\text{NumStars}$	0.037 (0.006)***	38.55***
AIC=35052.62; BIC=35133.00; $R^2_{\eta^2}=0.08$; $R^2_c=0.96$		
Num. obs.=3579; Num. groups: ProjectID=50		
*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$		
Log-transformed and standardized variables following Sec. IV-C.		

[52]. However, the above direction of correlation inverses at a high level of fork entropy. From the perspective of varying fork entropy by fixing the other factor, i.e., comparing the left and right parts separated by the vertical dashed line in Fig. 3(b), the acceptance rate generally increases with an increasing fork entropy with respect to the number of pull-requests contain test cases. But when the majority of pull-requests contain test cases, as shown by the part to the right of the dashed line, the increase in fork entropy instead shows a negative correlation with the acceptance rate.

We omit the details about the interactions between fork entropy and other control variables including NumFiles, ProjectAge, and ProjectAge, which show significant and negative interactions as listed in Table II due to page limites.

In summary, our answer to **RQ2** is as follows. *Fork entropy has a statistically significant, positive correlation with the acceptance rate of external pull-requests, albeit with a limited effect. Our analysis indicates that fork entropy also positive interacts with the number of pull-requests that touch hot files on the response. We find that the acceptance rate of pull-requests touching hot files only exhibits an upward trend when fork entropy is at a high level.*

C. **RQ3: What is the correlation between fork entropy and OSS projects' number of reported bugs?**

In this section, we study the correlation between fork entropy and the number of reported bugs of OSS projects. We model the number of bug-report issues as a function of fork entropy with controlling other factors including the number of forks, the number of modified files, project age, and the number of stars. Interactions between fork entropy and each controlled factor are also involved in the model as fixed effects. Table III summarizes the results of the number of reported bugs model. The model fits the data well by explaining about 96% of the variability by including both fixed and random effects, achieving a considerable improvement compared to the fixed effects alone by about 88 percentage points.

The results in Table III suggest that fork entropy significantly and negatively correlates to the number of bug-report issues with an estimated coefficient of -0.086. The absolute

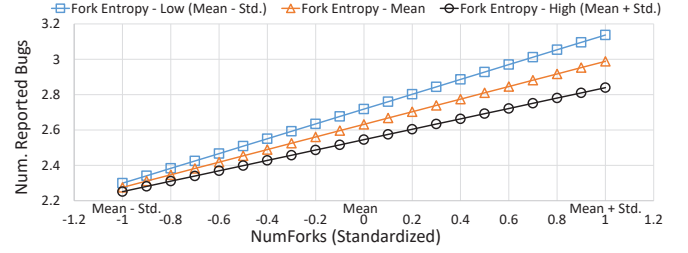


Fig. 4. The interaction between fork entropy and NumForks in the number of reported bugs model shown in Table III.

value is the third largest among the factors, lower than the estimated coefficient of NumForks and ProjectAge, which are 0.356, and 0.115, respectively. The results also suggest that the number of modified files significantly and negatively correlates to the number of bug-reporting issues, while the other controlled factors show significant and positive correlations with the response. All interaction terms show significance, with the interaction of fork entropy and the number of forks negatively correlates to the response.

We observe an opposition between fork entropy and the number of forks, where the former negatively correlates to the response and the latter positively relates to the response, respectively. Fig. 4 shows the interaction. We find that more bug-report issues are submitted with the increasing in the number of forks regardless of fork entropy. There is a slower growth in bug-report issues when fork entropy is at a higher level. Combined with the results in **RQ 1** and **RQ 2**, a possible explanation to the above result is that increased fork entropy are related to more commits and pull-requests been accepted, including bug-fixing ones, before potential bugs are reported. With the above results, if we agree with the assumption that less reported bugs indicates a higher software quality [26], [27], we can state that a higher level of fork entropy is positively linked to an improved software quality.

In summary, we answer **RQ3** as below. *Our study reveals a significant, negative correlation between fork entropy and the number of reported bugs in OSS projects. Furthermore, it shows that the rising trend of bug-reporting issues associated with the increasing number of OSS projects' forks is reduced when fork entropy is at high levels.*

VI. DISCUSSION

This section discusses the implications of our work, and the threats to validity.

A. Implications and Discussion

This section discuss the implications of our work.

Developing metrics and models that measure the health of OSS projects has received much attention from both researchers and practitioners of OSS recently. For example, communities such as CHAOSS [67] and OSS Compass [54] are founded to achieve an understanding, and provide services to measure the health of OSS communities, projects, and ecosystems through numerous qualitative and quantitative

metrics. Fork related metrics, such as the number of technical forks [68], is considered as a useful indicator. The fork entropy proposed in this paper is a new metric about an OSS project's forks, and can potentially be added to the collection of metrics provided by the above communities due to its correlations with projects' external productivity, pull-request acceptance ratio, and number of reported bugs as shown in our studies.

The proposed fork entropy metric also provides opportunities for further research on OSS projects' forks. For instance, one could utilize pattern mining and time series analysis technologies to examine the evolving patterns and future trends of fork entropy over time [69]. Furthermore, it is potentially important to identify and assess the various factors and events that contribute to the rise and fall of fork diversity in OSS projects, and analyze their impact on the sustainability and prosperity of OSS projects [12], [33]. With a thorough understanding of fork diversity trends and influencing factors, it is possible to develop monitoring tools and guidelines to facilitate effective forking and collaboration during the development and maintenance of OSS projects.

B. Threats to Validity

We discuss the threats to validity as follows.

Construct Validity. This work explores the diversity of OSS projects' fork populations measured by the proposed fork entropy metric. The construct validity concerns about whether the fork entropy measures the diversity of the fork population. Because fork entropy is built on top of the well-established metric of Rao's quadratic entropy [9] which has shown to be effective in measuring population diversity [10], and because we have shown the properties of fork entropy including symmetry, continuity, and monotonicity, we argue that the proposed fork entropy is valid in measuring the diversity of an OSS project's fork population.

Internal Validity. In this work, we conduct a data-driven approach to study the correlation between fork entropy and OSS projects' external productivity, pull-request acceptance rate, and number of reported bugs. We also restrict our scope to forks owned, and contributions made, by external contributors. The restriction is designed to make our study more focused on fork-related properties about OSS projects. However, there are many other factors affecting the contributions made to an OSS project [13]. External contributors' contribution are also partially determined by the core-members of OSS teams. As a result, the conclusions made in this paper do not fully explain how fork-related contributions made to OSS projects or imply a causal relationship. In addition, we set the time interval to one month when taking snapshots to build file modification matrices and calculate fork entropy. Although this size is widely used in previous OSS-related studies [26], changing the time granularity during the analysis may potentially change the results. It is our future work to test the results with different time intervals.

External Validity. We select fifty projects from GitHub that cover various application domains in our studies. However, the collection of projects studied is small compared to the number

of projects hosted on OSS platforms. The conclusions made in this paper may not generalize well to other projects, or to projects hosted on other platforms. It is our future work to include more projects in our studies.

VII. CONCLUSION

In this work, we focus on the pull-based OSS development and propose a novel metric called fork entropy to measure the diversity of the population of forks around an OSS project. We calculate fork entropy by applying the Rao's quadratic entropy with a distance function that measures the dissimilarity of the forks' modifications to project files. By conducting empirical studies on fifty real-world OSS projects from GitHub, we reveal that there exist significant correlations between fork entropy and the external productivity, the acceptance rate of external pull-requests, and number of reported bugs in these projects. We also find significant interactions between fork entropy and other influencing factors such as the number of forks, project age, ratio of pull-requests that touch hot files, etc, which have previously found to correlate to the productivity and quality of OSS projects. The proposed fork entropy metric not only enriches the current available metrics about understanding OSS projects' forks, it also offers opportunities for conducting further research on pull-based social coding.

ACKNOWLEDGMENT

We thank OSS Compass and GHTorrent in helping us to obtain data about OSS projects' forks. This work is supported by NSFC No. 62172203, Fundamental Research Funds for the Central Universities, and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

REFERENCES

- [1] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang, "Why and how developers fork what from whom in github," *Empirical Software Engineering*, vol. 22, no. 1, pp. 547–578, 2017.
- [2] S. Chacon and B. Straub, *Pro git*. Springer Nature, 2014.
- [3] Z. Li, Y. Yu, M. Zhou, T. Wang, G. Yin, L. Lan, and H. Wang, "Redundancy, context, and preference: An empirical study of duplicate pull requests in oss projects," *IEEE Transactions on Software Engineering*, 2020.
- [4] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 345–355.
- [5] I. Mergel, "Open collaboration in the public sector: The case of social coding on github," *Government Information Quarterly*, vol. 32, no. 4, pp. 464–472, 2015.
- [6] S. Zhou, B. Vasilescu, and C. Kästner, "What the fork: a study of inefficient and efficient forking practices in social coding," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 350–361.
- [7] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on computer supported cooperative work*, 2012, pp. 1277–1286.
- [8] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 805–816.
- [9] C. R. Rao, "Diversity and dissimilarity coefficients: a unified approach," *Theoretical population biology*, vol. 21, no. 1, pp. 24–43, 1982.

- [10] Z. Botta-Dukát, "Rao's quadratic entropy as a measure of functional diversity based on multiple traits," *Journal of vegetation science*, vol. 16, no. 5, pp. 533–540, 2005.
- [11] A. R. Hughes, B. D. Inouye, M. T. Johnson, N. Underwood, and M. Vellend, "Ecological consequences of genetic diversity," *Ecology letters*, vol. 11, no. 6, pp. 609–623, 2008.
- [12] U. Raja and M. J. Tretter, "Defining and evaluating a measure of open source project survivability," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 163–174, 2012.
- [13] S. Jansen, "Measuring the health of open source software ecosystems: Beyond the scope of project health," *Information and Software Technology*, vol. 56, no. 11, pp. 1508–1519, 2014.
- [14] J. Aué, M. Haisma, K. F. Tómasdóttir, and A. Bacchelli, "Social diversity and growth levels of open source software projects on github," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2016, pp. 1–6.
- [15] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, 2015, pp. 3789–3798.
- [16] G. Catolino, F. Palomba, D. A. Tamburri, A. Serebrenik, and F. Ferrucci, "Gender diversity and women in software teams: How do they affect community smells?" in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE, 2019, pp. 11–20.
- [17] S. Daniel, R. Agarwal, and K. J. Stewart, "The effects of diversity in global, distributed collectives: A study of open source project success," *Information Systems Research*, vol. 24, no. 2, pp. 312–333, 2013.
- [18] B. Vasilescu, V. Filkov, and A. Serebrenik, "Perceptions of diversity on git hub: A user survey," in *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE, 2015, pp. 50–56.
- [19] B. Vasilescu, A. Serebrenik, and M. G. van den Brand, "The babel of software development: Linguistic diversity in open source," in *International Conference on Social Informatics*. Springer, 2013, pp. 391–404.
- [20] L. Ren, S. Zhou, C. Kästner, and A. Wasowski, "Identifying redundancies in fork-based development," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 230–241.
- [21] S. Zhou, S. Stanculescu, O. Leßenich, Y. Xiong, A. Wasowski, and C. Kästner, "Identifying features in forks," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 105–116.
- [22] R. Padhye, S. Mani, and V. S. Sinha, "A study of external community contribution to open-source projects on github," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 332–335.
- [23] R. Krishnamurthy, V. Jacob, S. Radhakrishnan, and K. Dogan, "Peripheral developer participation in open source projects: an empirical analysis," *ACM Transactions on Management Information Systems (TMIS)*, vol. 6, no. 4, pp. 1–31, 2016.
- [24] G. Pinto, I. Steinmacher, L. F. Dias, and M. Gerosa, "On the challenges of open-sourcing proprietary software projects," *Empirical Software Engineering*, vol. 23, pp. 3221–3247, 2018.
- [25] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *2013 International Conference on Social Computing*. IEEE, 2013, pp. 188–195.
- [26] Z. Wang, Y. Feng, Y. Wang, J. A. Jones, and D. Redmiles, "Unveiling elite developers' activities in open source projects," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 3, pp. 1–35, 2020.
- [27] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of mozilla firefox," in *2012 9th IEEE working conference on mining software repositories (MSR)*. IEEE, 2012, pp. 179–188.
- [28] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 155–165.
- [29] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, "Impact of developer turnover on quality in open-source software," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 829–841.
- [30] Ş. Stănculescu, S. Schulze, and A. Wasowski, "Forked and integrated variants in an open-source firmware project," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 151–160.
- [31] M. Hadian, S. Ghari, M. Fokaefs, S. Brisson, E. Noei, K. Lyons, B. Adams, and S. Zhou, "Exploring trends and practices of forks in open-source software repositories," in *Proceedings of the 32nd Annual International Conference on Computer Science and Software Engineering*, 2022, pp. 120–129.
- [32] S. Zhou, B. Vasilescu, and C. Kästner, "How has forking changed in the last 20 years? a study of hard forks on github," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 445–456.
- [33] A. Rastogi and N. Nagappan, "Forking and the sustainability of the developer community participation—an empirical investigation on outcomes and reasons," in *2016 IEEE 23rd international conference on software analysis, evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 102–111.
- [34] J. Businge, M. Openja, S. Nadi, and T. Berger, "Reuse and maintenance practices among divergent forks in three software ecosystems," *Empirical Software Engineering*, vol. 27, no. 2, p. 54, 2022.
- [35] S. Brisson, E. Noei, and K. Lyons, "We are family: analyzing communication in github software repositories and their forks," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 59–69.
- [36] D. Imamura, T. Ishio, R. G. Kula, and K. Matsumoto, "Bug-fix variants: Visualizing unique source code changes across github forks," in *2022 Working Conference on Software Visualization (VISSOFT)*. IEEE, 2022, pp. 157–161.
- [37] V. S. Sinha, S. Mani, and S. Sinha, "Entering the circle of trust: developer initiation as committers in open-source projects," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 133–142.
- [38] P. M. Blau, *Inequality and heterogeneity: A primitive theory of social structure*. Free Press New York, 1977, vol. 7.
- [39] P. D. Allison, "Measures of inequality," *American sociological review*, pp. 865–880, 1978.
- [40] L. Jost, "Entropy and diversity," *Oikos*, vol. 113, no. 2, pp. 363–375, 2006.
- [41] A. J. Daly, J. M. Baetens, and B. De Baets, "Ecological diversity: measuring the unmeasurable," *Mathematics*, vol. 6, no. 7, p. 119, 2018.
- [42] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, "Understanding free/open source software development processes," pp. 95–105, 2006.
- [43] M. Rupp, "Machine learning for quantum mechanics in a nutshell," *International Journal of Quantum Chemistry*, vol. 115, no. 16, pp. 1058–1073, 2015.
- [44] S. Fadel, S. Ghoniemy, M. Abdallah, H. A. Sorra, A. Ashour, and A. Ansary, "Investigating the effect of different kernel functions on the performance of svm for recognizing arabic characters," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 1, pp. 446–450, 2016.
- [45] A. Alali, H. Kagdi, and J. I. Maletic, "What's a typical commit? a characterization of open source software repositories," in *2008 16th IEEE international conference on program comprehension*. IEEE, 2008, pp. 182–191.
- [46] O. Arafat and D. Riehle, "The commit size distribution of open source software," in *2009 42nd Hawaii International Conference on System Sciences*. IEEE, 2009, pp. 1–8.
- [47] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, "The space of developer productivity: There's more to it than you think," *Queue*, vol. 19, no. 1, pp. 20–48, 2021.
- [48] G. Murić, A. Abeliuk, K. Lerman, and E. Ferrara, "Collaboration drives individual productivity," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–24, 2019.
- [49] I. Steinmacher, G. Pinto, I. S. Wiese, and M. A. Gerosa, "Almost there: A study on quasi-contributors in open-source software projects," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 256–266.
- [50] R. Nadri, G. Rodriguez-Perez, and M. Nagappan, "Insights into non-merged pull requests in github: Is there evidence of bias based on perceptible race?" *IEEE Software*, vol. 38, no. 2, pp. 51–57, 2020.
- [51] P. Willett, "The porter stemming algorithm: then and now," *Program*, 2006.

- [52] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th international conference on Software engineering*, 2014, pp. 356–366.
- [53] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2487085.2487132>
- [54] "OSS Compass," <https://www.oss-compass.org>, 2023, [Online; accessed 16-August-2023].
- [55] C. E. Metz, "Basic principles of roc analysis," in *Seminars in nuclear medicine*, vol. 8, no. 4. Elsevier, 1978, pp. 283–298.
- [56] J. W. Osborne and A. Overbay, "The power of outliers (and why researchers should always check for them)," *Practical Assessment, Research, and Evaluation*, vol. 9, no. 1, p. 6, 2004.
- [57] N. E. Breslow and D. G. Clayton, "Approximate inference in generalized linear mixed models," *Journal of the American statistical Association*, vol. 88, no. 421, pp. 9–25, 1993.
- [58] D. Bates, M. Maechler, B. Bolker, and S. Walker, "Linear mixed-effects models using 'eigen' and s4 [r package lme4 version 1.1-11]," 2016.
- [59] A. Gelman and J. Hill, *Data analysis using regression and multi-level/hierarchical models*. Cambridge university press, 2006.
- [60] P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology press, 2014.
- [61] P. C. Johnson, "Extension of nakagawa & schielzeth's r2glmm to random slopes models," *Methods in ecology and evolution*, vol. 5, no. 9, pp. 944–946, 2014.
- [62] S. Nakagawa and H. Schielzeth, "A general and simple method for obtaining r^2 from generalized linear mixed-effects models," *Methods in ecology and evolution*, vol. 4, no. 2, pp. 133–142, 2013.
- [63] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Selected papers of hirotugu akaike*. Springer, 1998, pp. 199–213.
- [64] G. Schwarz, "Estimating the dimension of a model," *The annals of statistics*, pp. 461–464, 1978.
- [65] D. Lüdtke, M. S. Ben-Shachar, I. Patil, P. Waggoner, and D. Makowski, "performance: An R package for assessment, comparison and testing of statistical models," *Journal of Open Source Software*, vol. 6, no. 60, p. 3139, 2021.
- [66] L. S. Aiken, S. G. West, and R. R. Reno, *Multiple regression: Testing and interpreting interactions*. sage, 1991.
- [67] "CHAOSS," <https://chaoss.community/>, 2017, [Online; accessed 16-August-2023].
- [68] "Metric: Technical Fork," <https://chaoss.community/kb/metric-branch-lifecycle/>, 2017, [Online; accessed 16-August-2023].
- [69] A. Amin, L. Grunske, and A. Colman, "An automated approach to forecasting qos attributes based on linear and non-linear time series modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 130–139.