Université de Mons
Faculté des Sciences
Département d'Informatique

Improving Bot Identification in Collaborative Software Development on GitHub

Natarajan Chidambaram

A dissertation submitted in fulfilment of the requirements of the degree of *Docteur en Sciences*

May 22, 2025

Acknowledgements

As I receive my highest academic degree, I am grateful to the people who have supported me throughout my PhD journey. Your guidance, encouragement, assistance, and support have been invaluable in helping me reach this milestone.

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Dr. Tom Mens, for his unwavering support and guidance throughout this journey. His constant encouragement, insightful expertise, exceptional ideas and constructive feedback have been invaluable in shaping my research and academic growth. I am especially grateful for his belief in my abilities despite my limited knowledge in software engineering, and for providing me with the opportunity to work on this exciting and timely research topic. His insights and guidance have been instrumental in refining my ideas, conceptualising it and improving the quality of my work. I have learnt so much about research in general and teaching through practical sessions. The recommendation letters he wrote for me, from getting a student volunteer position at an international conference to my role as part of the organising team for multiple international scientific workshops have been instrumental in helping me to gain key experiences and skills. Also, thank you for your patience in waiting for me to provide the text, tables and figures in the last-minute and still finding time to provide reviews, suggestions, and corrections for me to obtain high-quality results and publications. I feel privileged to have had the opportunity to work with such a dedicated professor who motivated and pushed me beyond my expectations, and always being available whenever I required his valuable guidance. Last but not the least, I am very thankful for encouraging and allowing me to explore the world too (through seminars, workshops and conferences) right from Melbourne in Australia to Ottawa in Canada.

I would also like to extend my gratitude to my co-supervisor Prof. Dr. Souhaib Ben Taieb. Although I did not interact with him much during my PhD, I am grateful for the support, guidance and insights that he provided through short talks during coffee/lunch breaks.

I would also like to extend my thanks to the committee members, Prof. Dr. Stéphane Dupont, Prof. Dr. Gregorio Robles, and Prof. Dr. Mairieli Wessel for reading my year-end committee reports, and providing their valuable feedback on improving my work. Their constructive criticism during the committee meetings has helped me improve the quality of this thesis. Also, I would like to thank Prof. Dr. Alexander Serebrenik for accepting to be part of my jury and for his valuable feedback on my work.

I would like to extend my sincere thanks to Dr. Alexandre Decan, for his assistance, feedback and support throughout my PhD. It was very crucial in helping me to conceptualise the ideas, refine and improve the quality of my research, writing, coding, and providing replication packages. His knowledge and expertise in the field of software engineering and data science have been invaluable in shaping my research. I am grateful for his willingness to share his knowledge and insights through long discussions, and for the support he provided me to navigate the challenges of my PhD journey. Also, a special thanks for improving the text in our publications.

I am grateful to my colleagues at the University of Mons, especially the members of the software engineering lab, for their support and camaraderie. Also, I would like to thank my colleagues from TRAIL, with whom I had a great opportunity to collaborate, write project proposals, create a plan for executing research projects, work and learn. I would like to acknowledge the support of the Walloon region and the DigitalWallonia4.AI initiative for providing funding for my research. Their commitment to promoting research and innovation has made this work possible.

I would like to thank all my friends who, one way or another, have supported me during my PhD. I am grateful for the moments we shared, the laughter, and the support you provided during challenging times. Your presence and encouragement have made this journey more enjoyable and fulfilling.

I would like to express my immense gratitude to my parents, who have been a constant source of support and encouragement throughout my life, including these past years during my PhD. Their love and belief in me have been a constant source of motivation. This achievement wouldn't have been possible if not for their availability for my untimely virtual calls, willingness to listen to me and provided guidance whenever I required. Also, I am thankful to my brother for listening to my technical talks over the phone and providing me with the much-needed support and encouragement. I would like to express my heartfelt gratitude to my wife, who came into my life when I started writing my thesis, and understanding me throughout the last and important phase of this journey.

Abstract

Contemporary social coding platforms such as GitHub facilitate collaborative distributed software development. This enables developers to contribute to software project repositories from different parts of the world. Developers engaged in this platform often perform various activities such as committing files, creating pull requests, performing code reviews, creating and deleting branches, updating documentation, deploying and releasing new software versions and so on. As these activities could be effort-intensive, repetitive and error-prone, repository maintainers and developers frequently use automated mechanisms (e.g., bot accounts, GitHub Apps, GitHub automated workflows, and other internal or external automation services) for performing these activities. Bot accounts and GitHub Apps are being widely used in GitHub repositories and are among the most active contributors in certain repositories. Determining whether a contributor corresponds to a bot or a human is important in socio-technical studies, for example to assess the positive and negative impact of using a bot, analyse their evolution and usage, identify and accredit top contributors, and so on.

The main aim of this dissertation is to improve bot identification in GitHub. While multiple bot identification approaches have been proposed in the past, they suffer from certain limitations that make them difficult to be used in practice. By creating multiple datasets, developing new bot identification models, and performing quantitative analysis, we provide several novel contributions. We show that bots are regularly among the most active contributors, although GitHub does not explicitly acknowledge the presence of several bots. Also, we show that existing bot identification approaches do not perform well in identifying bot contributors. As a first step, we develop two models to improve bot identification in GitHub. One model leverages the predictions made by an existing approach across multiple GitHub repositories and provide an overall improved performance. Another is an ensemble model that combines the prediction made by these existing approaches to improve bot identification in GitHub. Then, we propose a dataset of contributor activity sequences that can be extracted from low-level events provided by the GitHub REST API. Based on these activities, we identify features that can statistically differentiate bots from human contributors engaged in collaborative software development. Also, we propose a manually labelled ground-truth of bots and human contributors that can be used to compare the performance, efficiency and limitations of existing bot identification approaches. Using this ground-truth and distinguishing features, we train BIMBAS, a new binary classification model to identify bots based on their recent activities in GitHub. Through an empirical study, we use BIMBAS to detect the presence of bots among hundreds of contributors and identify how bots are being used in a large software ecosystem. We reveal behavioural differences between bots and humans, and between different bot categories. To enable the practical use of BIMBAS, we develop an open-source bot identification tool called RABBIT, which outperforms the efficiency of existing bot identification approaches.

Contents

1 Introduction 1
1.1 Collaborative software development 2
1.2 Automation practices in GitHub 3
1.3 GitHub contributors 4
1.4 Prevalence of bots in large GitHub repositories 8
1.5 Thesis statement and goals 10
1.6 Thesis structure 12

2 Background 17
2.1 GitHub automation mechanisms 18
2.2 GitHub Actions workflows 19
2.3 Internal automation services 21
2.4 GitHub Apps 22
2.5 Bot accounts 23
2.6 External automation services 25
2.7 GitHub REST API 26
2.8 Summary 29

3 State of the art 31
3.1 Studies on bots 32
3.1.1 Categorising bots 32
3.1.2 Challenges and impact of bots 33
3.1.3 Empirical studies 35
3.2 Bot identification in GitHub 37
3.2.1 Approaches to identify bots 37
3.2.2 Studies that relied on bot identification approaches 39

4 Improving existing bot identification approaches 41
4.1 An ensemble bot identification model 42
4.1.1 Motivation 42
4.1.2 Ground-truth dataset of contributors 43

4.1.3 Ensemble classification model 43
4.1.4 Prevalence of bots among active contributors 46
4.1.5 Presence of bots in commits 47
4.2 Leveraging predictions in multiple repositories 48
4.2.1 Motivation 48
4.2.2 Dataset 49
4.2.3 Contributor activeness in multiple repositories 50
4.2.4 Number of different or incomplete predictions 51
4.2.5 Leveraging different predictions 51
4.2.6 Leveraging incomplete predictions 53
4.3 Summary and conclusions 54

5 Differentiating bots from humans based on activities 55
5.1 Identifying activities from GitHub events 56
5.1.1 Motivation and structure 56
5.1.2 Dataset construction 56
5.1.3 Data schema 60
5.1.4 Limitations 61
5.2 Preliminary features differentiating bots and humans 62
5.2.1 Curated Dataset 62
5.2.2 Statistical methods 63
5.2.3 Number of activity types 64
5.2.4 Specialisation of activity types across repositories 65
5.2.5 Number of repositories 66
5.2.6 Time to switch between repositories 68
5.2.7 Time between consecutive activities 69
5.3 Summary and conclusions 70

6 An activity-based bot identification model 73
6.1 Motivation and objectives 74
6.2 Creating a ground-truth dataset of bots and humans 75
6.3 Performance limitations of existing bot identification approaches . . . 76
6.3.1 Excluded bot identification approaches 76
6.3.2 Experimental setup 77
6.3.3 Name-Based Heuristic 78
6.3.4 BoDeGHa 78
6.3.5 BoDeGiC 79
6.3.6 BotHunter 79
6.3.7 Summary 80
6.4 Creating the BIMBAS bot identification model 80
6.4.1 Extracting activity sequences 81
6.4.2 Selecting features 81
6.4.3 Model selection 82
6.4.4 Training and evaluating BIMBAS 85
6.4.5 Feature importance 87

6.4.6 Analysing the misclassifications 87
6.5 Threats to validity 89
6.6 Summary 90

7 Using BIMBAS in practice 91
7.1 A large scale analysis with BIMBAS 92
7.1.1 Extracting activity sequences for contributors 92
7.1.2 Bot detection 93
7.1.3 Prevalence of bots in *NumFOCUS* repositories and organisations 94
7.1.4 Which activity types are contributors involved in? 96
7.1.5 Which organisations are contributors involved in? 99
7.2 An efficient bot identification tool 101
7.2.1 Implementation of RABBIT 102
7.2.2 Comparing RABBIT's efficiency with existing approaches 104
7.2.3 Limitations 105
7.3 Summary and conclusions 105

8 Conclusion 107
8.1 Contributions 108
8.1.1 Datasets created 108
8.1.2 Bot identification models 108
8.1.3 Tool development 110
8.1.4 Empirical analysis 111
8.2 Limitations and discussions 112
8.2.1 GitHub API 112
8.2.2 Methodological limitations 112
8.3 Perspectives 114
8.3.1 Datasets 114
8.3.2 Bot identification approaches 115
8.3.3 Empirical analyses 116

CHAPTER 1

Introduction

I started my PhD in August 2021 in the software engineering lab at the University of Mons, Belgium. This thesis provides details on the empirical analyses that were performed, and the machine learning models and tools that were developed to improve identification of bot contributors in GitHub. This thesis is supervised by Prof. Tom Mens and Prof. Souhaib Ben Taieb.

Multiple software developers often join to develop complex software applications in a collaborative manner. Such collaboration practices have become a cornerstone of modern software development. In this chapter, we briefly discuss collaborative software development in GitHub and various automation practices in GitHub. We analyse the prevalence of automation mechanisms in GitHub repositories, distinguish different type of contributors in GitHub, and detail the objectives of this thesis.

1.1. Collaborative software development

The software development process involves identifying functional and non-functional requirements, designing and implementing software applications, testing them, creating documentation, releasing and deploying the software and maintaining it throughout its lifetime. Each of these steps is crucial for a software project to serve its users community for a long time. To achieve this, software development often requires concerted and collaborative effort from multiple developers that work together as a team.

The demand for better software products, capitalising on the global resource pool, providing faster services, and round-the-clock development has led software development businesses to distribute their development processes overseas, also known as global software development (Herbsleb & Moitra, 2001). This has made software development an increasingly multisite, multicultural and globally distributed undertaking. It is usual for software companies to have their teams of developers geographically dispersed across different time zones, often on more than one continent (Herbsleb, 2007). Companies adopting global software distribution practices often face many challenges related to socio-cultural distance, temporal distance and geographical distance (Holmstrom et al., 2006). The emergence of social coding platforms such as GitHub, GitLab, Gitea and BitBucket has solved some of these problems by creating and sustaining a coherent connection among distributed individuals occupying a shared cloud space. By creating separate feature branches in software projects, developers can have a dedicated workspace and merge their code frequently through pull requests (PR) (Riehle et al., 2009). This enables contributions from third-party developers as well. Collaborative software development practices have become widespread in the last decade (Costa et al., 2011; Dabbish et al., 2012) with a sharp rise in popularity of open-source software (OSS) development among big organizations (McClean et al., 2021; Butler et al., 2022) and the emergence of social coding platforms.

GitHub is the biggest social coding platform with over 100 million developers[1] and more than 420 million repositories as of January 2023. It allows contributions in various forms such as creating new issues when someone detects a bug or a security problem, wants to have a new feature, would like to improve the quality of code, wishes to improve documentation and so on. Another example of collaboration is through PRs (Arora et al., 2017) to integrate new contributions to the software. After performing code reviews, the project maintainers take the necessary steps (Tsay et al., 2014) to include or reject the requested contribution. Also, the project maintainers might communicate with the contributors through discussions using GitHub Discussion pages (Hata et al., 2021), comments under issues (Kavaler et al., 2017), PRs (Zhang et al., 2023) and so on. Additionally, developers and project maintainers communicate through other channels that integrate with GitHub. For example, through Stack Overflow for technical questions and answers,[2] Slack,[3] Discord,[4] Tele-

[1] `https://github.blog/news-insights/company-news/100-million-developers-and-counting/`
[2] `https://github.com/marketplace/stack-overflow-extension-for-github-copilot`
[3] `https://github.com/marketplace/slack-github`
[4] `https://github.com/marketplace/actions/discord-courier`

gram, WhatsApp, Mastodon and Gitter for communication among team members, and JIRA[5] for tracking issues.

1.2. Automation practices in GitHub

Repositories hosted on GitHub allow contributions in various forms for a wide variety of activities (e.g., coding, documenting, checking dependency and fixing security vulnerabilities) that developers and maintainers can perform. Many GitHub repositories allow contributors to submit new code through PRs. This enables project maintainers to discuss and evaluate external contributions. However, this pull-based development method significantly increases the workload of project repository maintainers (Gousios et al., 2015) as the code has to be reviewed, tested, verified with respect to corresponding documentation and adhere to contributor license agreements and so on. For small projects, maintainers are able to handle the incoming contributions as their number may remain manageable, but larger projects may face difficulties to scale and keep up with the pace of maintaining high quality software releases and managing their dependencies (Young et al., 2021).

As performing such activities can be repetitive, error-prone and effort-intensive, developers and maintainers tend to adopt automation practices in GitHub. Automation in GitHub repositories can be achieved through various mechanisms:

- GitHub workflows (Wessel et al., 2023a) are configure automated processes adopted in repositories to perform one or more jobs.
- GitHub Apps[6] are specialised tools that integrate with GitHub, extend GitHub's functionality and interact with it.
- GitHub internal automation services (Rebatchi et al., 2024) are readily available for use by GitHub in repositories by enabling or disabling them in the repository settings (e.g., `Dependabot` for automatic dependency updates).
- Bot accounts (Beschastnikh et al., 2017) are GitHub user accounts that perform automated tasks in repositories.
- External automation services (Biehl, 2017) are third-party tools or systems that interact with GitHub through webhooks and APIs to automate tasks (e.g., `Jenkins` for continuous integration, JIRA for issue tracking, `SonarQube Cloud` and `Codacy` for code quality analysis, `Snyk` for scanning vulnerabilities and so on).

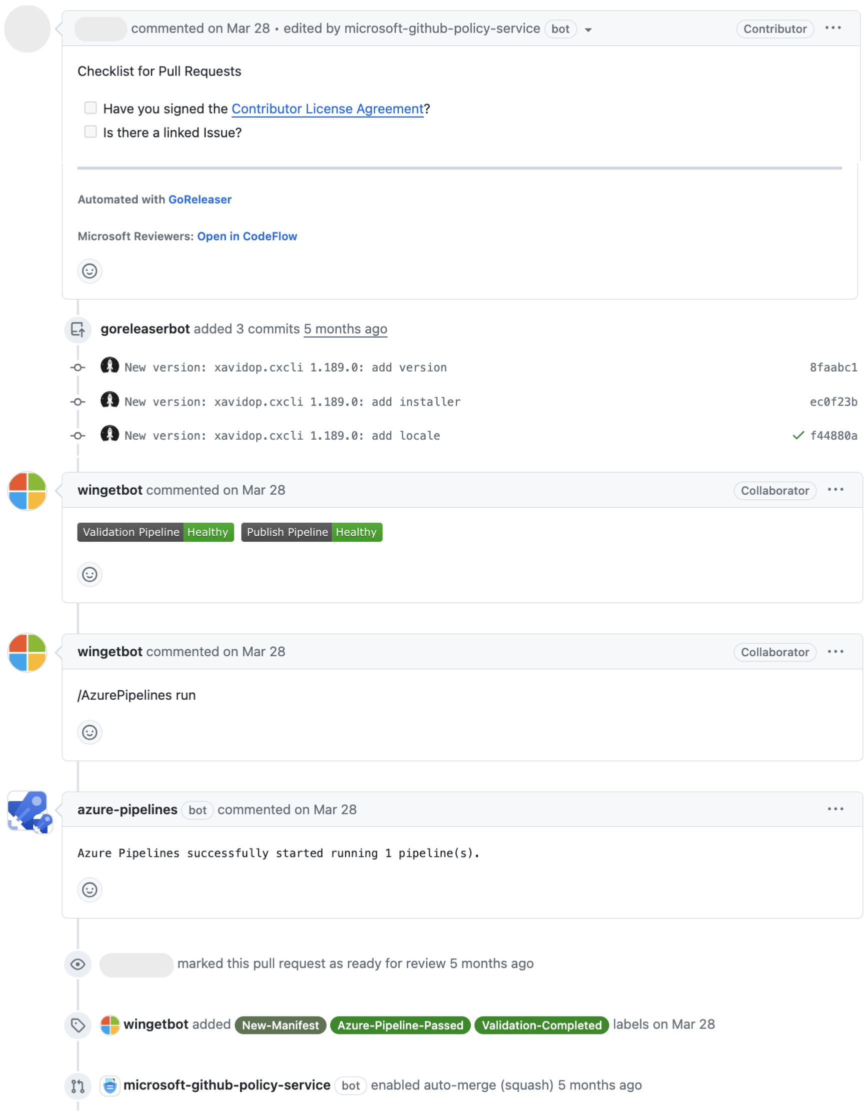
A detailed explanation of these automation practices will be provided in Chapter 2.

[5] `https://github.com/marketplace/actions/setup-jira`

[6] `https://docs.github.com/en/apps/creating-github-apps/about-creating-github-apps/about-creating-github-apps`

1.3. GitHub contributors

The previous section mentioned various automation practices that exist on GitHub. These automation mechanisms contribute to the project development in GitHub repositories. This section explains the different types of contributors that are considered in this dissertation.


commented on Mar 28 • edited by microsoft-github-policy-service bot Contributor

Checklist for Pull Requests

- [ ] Have you signed the Contributor License Agreement?
- [ ] Is there a linked Issue?

Automated with GoReleaser

Microsoft Reviewers: Open in CodeFlow

goreleaserbot added 3 commits 5 months ago

```
New version: xavidop.cxcli 1.189.0: add version      8faabc1
New version: xavidop.cxcli 1.189.0: add installer    ec0f23b
New version: xavidop.cxcli 1.189.0: add locale     ✓ f44880a
```

wingetbot commented on Mar 28 Collaborator

Validation Pipeline Healthy Publish Pipeline Healthy

wingetbot commented on Mar 28 Collaborator

/AzurePipelines run

azure-pipelines bot commented on Mar 28

```
Azure Pipelines successfully started running 1 pipeline(s).
```

marked this pull request as ready for review 5 months ago

wingetbot added New-Manifest Azure-Pipeline-Passed Validation-Completed labels on Mar 28

microsoft-github-policy-service bot enabled auto-merge (squash) 5 months ago

Figure 1.1: An interaction between a human account, bot account and a bot actor corresponding to an App in a PR in *microsoft/winget-pkgs* repository.

According to the GitHub documentation,[7] an *actor* is an object (typically a *User* or a *Bot*) that can perform activities in GitHub. For the bot actors that correspond to GitHub Apps and GitHub internal automation services, the GitHub REST API's `users` endpoint[8] (which we refer to as `users` endpoint for short) reports their type as *Bot*, while bot accounts and personal user accounts are reported as being of type *User*. To carry out activities on a GitHub repository, a human contributor needs to have a *personal user account*, or an *organisation account*.[9] Every GitHub user signs in with their personal account to use GitHub, whereas an organisation account enhances collaboration between multiple personal accounts. Organisation accounts will not be considered in this thesis, they serve a different purpose and are therefore considered out of scope for our research. In the GitHub UI, an activity performed by a *User* will be visible under their login name (a unique identifier of a user account in GitHub).

As an illustration, Fig. 1.1 shows a PR[10] created in *microsoft/winget-pkgs* repository by a human account (its account name has been greyed out for privacy purposes). One can observe a couple of comments by the (wingetbot) bot account. This bot account automates various processes such as automatically updating dependencies and creating PRs, automating processes in a PR once it is created, and so on. In Fig. 1.1, wingetbot automatically checks the quality of the PR (verifying if the pipeline is working as expected), invokes the azure-pipelines[bot] bot actor for the GitHub App `Azure Pipelines`[11] (a GitHub App that continuously integrates, tests and deploys software changes) with the required input and add labels upon successful execution of this pipeline.

An activity performed by a GitHub App (such as `Azure Pipelines`) or a GitHub internal automation service (such as `Dependabot`) will be visible in the GitHub UI under the name of its respective bot actor. For example, in Fig. 1.1, `Azure Pipelines` uses its bot actor azure-pipelines[bot] to make a comment "Azure Pipelines successfully started running 1 pipeline(s)" in this PR. Another bot actor that can be observed in Fig. 1.1 is microsoft-github-policy-service[bot] which corresponds to the `Microsoft GitHub Policy Service` GitHub App. It is created and maintained by the GitHub inside Microsoft team, and is the foundation for the compliance and governance efforts on their GitHub instances.

Fig. 1.2 shows another example. Whenever a PR is created in *tommens/calculator-cucumber*, a GitHub Actions automated workflow[12] is executed, and its bot actor, github-actions[bot], posts a comment "Thanks for reporting!" in this PR. The figure also illustrates the use of `SonarCloud` that detects security vulnerabilities, bugs and code smells and provides remediation guidance to help fix issues in the code. A webhook to `SonarCloud` is triggered upon PR creation, resulting in a PR comment

[7] `https://docs.github.com/en/graphql/reference/interfaces#actor`

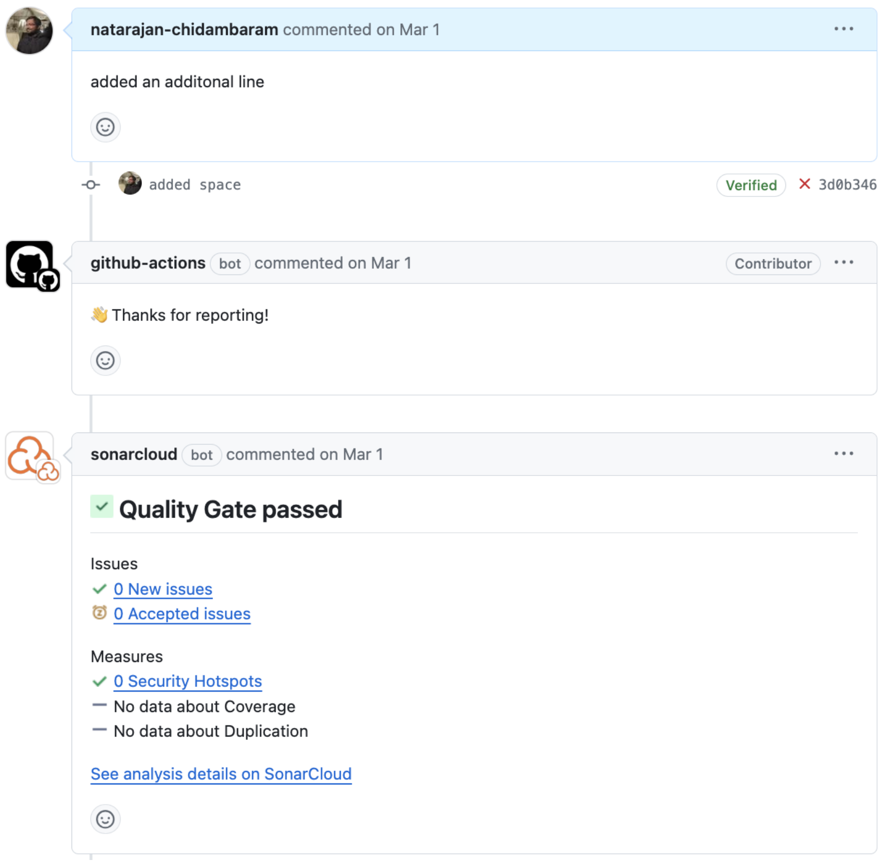
[8] `https://api.github.com/users`

[9] `https://docs.github.com/en/get-started/learning-about-github/types-of-github-accounts`

[10] `https://github.com/microsoft/winget-pkgs/pull/146788`

[11] `https://github.com/apps/azure-pipelines`

[12] `https://github.com/tommens/calculator-cucumber/blob/master/.github/workflows/comment.yml`



natarajan-chidambaram commented on Mar 1

added an additonal line

added space — Verified — 3d0b346

github-actions bot commented on Mar 1 — Contributor

👋 Thanks for reporting!

sonarcloud bot commented on Mar 1

✓ **Quality Gate passed**

Issues
✓ 0 New issues
0 Accepted issues

Measures
✓ 0 Security Hotspots
— No data about Coverage
— No data about Duplication

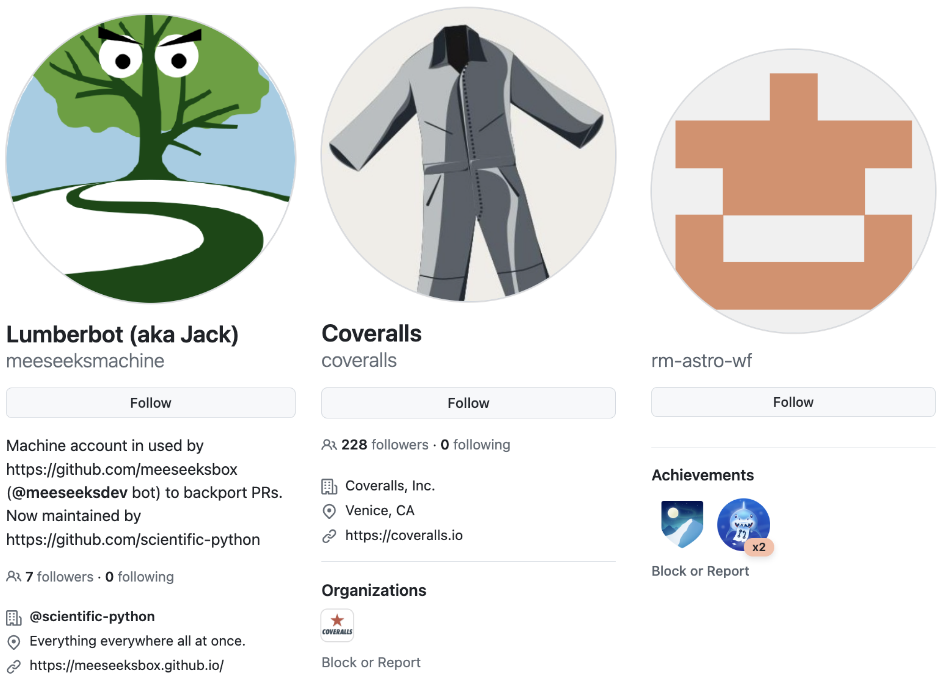
See analysis details on SonarCloud

Figure 1.2: A PR in *tommens/calculator-cucumber* repository is commented by the respective bot actor for the internal automation service for GitHub Actions workflows as well as the external quality analysis service - *SonarCloud*.

using the `SonarCloud` GitHub App[13] that uses its bot actor sonarcloud[bot] to post the comment.

GitHub also allows the use of so-called *machine users* to automate activities that are repetitive and error-prone. A machine user has a personal account just like a human, but does not always reveal itself as an automated tool in its GitHub account profile. For example, Fig. 1.3 shows three different account profiles of machine users. Fig. 1.3a explicitly signals itself as a machine account and provides details such as a link to its website, purpose and so on. Fig. 1.3b shows the machine user coveralls for providing automated test coverage reports that does not immediately reveals itself as a tool, but it does provide a link to the tool's website and location. Fig. 1.3c shows a machine user for merging PRs that does not reveal any information about

[13]`https://github.com/marketplace/sonarcloud`


Lumberbot (aka Jack)
meeseeksmachine
Follow
Machine account in used by https://github.com/meeseeksbox (@meeseeksdev bot) to backport PRs. Now maintained by https://github.com/scientific-python
7 followers · 0 following
@scientific-python
Everything everywhere all at once.
https://meeseeksbox.github.io/

Coveralls
coveralls
Follow
228 followers · 0 following
Coveralls, Inc.
Venice, CA
https://coveralls.io
Organizations
Block or Report

rm-astro-wf
Follow
Achievements
x2
Block or Report

(a) (b) (c)

Figure 1.3: Example of a machine user that (a) reveals itself as a machine account in its profile description and has a link to its website, (b) does not have a profile description but has a link to its website, and (c) does not reveal any information.

itself making it difficult to know if we are dealing with a human account or a machine user. Detecting machine users is crucial for socio-technical studies of collaborative software development that require the activities performed by machine users to be treated differently from that of humans.

This dissertation will adopt the following terminology regarding GitHub contributor types. We will refer to human GitHub contributors having a personal user account as **human accounts**, to the machine users as **bot accounts**. GitHub also allows using internal automation services (e.g., `Dependabot`) and so-called **Apps**[14] in repositories to perform automated activities. We will collectively refer to these two automation mechanisms as **bot actors**. We will use the term **bots** to collectively refer to both bot accounts and bot actors. We will use the term **humans** to refer to human accounts. We will use the generic term **contributors** to collectively refer to both humans and bots. Fig. 1.4 visually summarises this terminology.

[14] `https://docs.github.com/en/apps`

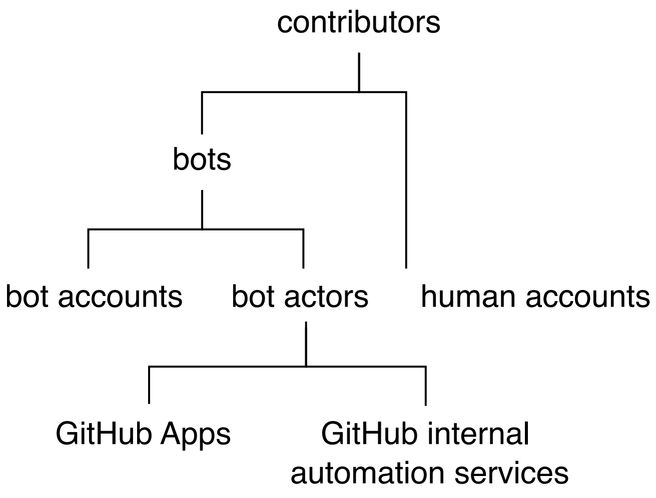


Figure 1.4: Terminology used in this dissertation.

1.4. Prevalence of bots in large GitHub repositories

As GitHub repositories can be quite large in terms of number of contributors and their activities, bots tend to be frequently used to automate effort-intensive, repetitive and error-prone activities. They perform a wide range of tasks including refactoring, generating bug patches, updating dependencies, and checking licenses (Wyrich & Bogner, 2019; Monperrus et al., 2019; Mirhosseini & Parnin, 2017). To demonstrate the prevalence of bots in GitHub repositories, we performed a preliminary study in November 2021 (Golzadeh et al., 2022b) based on a selection of 10 large and active open-source projects for popular programming languages (JavaScript, Java, Python and Rust):

1. VueJS (`https://github.com/vuejs/vue`), a very popular front-end framework for JavaScript with more than 350 contributors and 3K commits.

2. Servo (`https://github.com/servo/servo`), an experimental browser engine written in Rust with more than 1K contributors, 40K commits and 30K dependents.

3. Cucumber JVM (`https://github.com/Cucumber/cucumber-jvm`), a Java implementation of the popular cucumber test framework, with more than 250 contributors and 6K commits.

4. Libc (`https://github.com/rust-lang/libc`), a Rust implementation to interoperate with "C-like" code on platforms supporting Rust. It has more than 500 contributors and 6K commits.

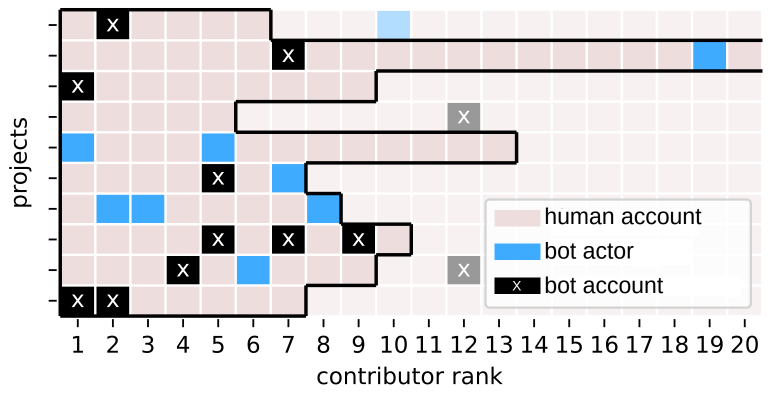


Figure 1.5: Bots among the top 20 most active committers in 10 popular open-source projects.

5. Boto (`https:github.com/boto/boto3`), an Amazon Web Services Software Development Kit for Python that has more than 5K commits, 100 contributors and 400K dependents.

6. Mockito (`https://github.com/mockito/mockito`), a popular Java mocking framework for unit tests with more than 250 contributors and 6K commits.

7. Rollup (`https://github.com/rollup/rollup`), a module bundler for JavaScript with more than 300 contributors, 5K commits and 11M dependents.

8. ts-jest (`https://github.com/kulshekhar/ts-jest`), a JavaScript testing framework that is used to test projects written in Typescript. It has 170 contributors, more than 4K commits and more than 1M dependents.

9. Prettier (`https://github.com/prettier/prettier`) is an opinionated code formatter written in JavaScript. It has more than 600 contributors with more than 9K commits and 7M+ dependents.

10. Karma (`https://github.com/karma-runner/karma`) is a test runner for JavaScript. It has more than 300 contributors, 2.5K commits and 3M dependents.

We retrieved the contributors with the highest number of commits in these ten projects, and determined their contributor type. To do so, we relied on the `users` endpoint. We retrieved the type (i.e., *User* or *Bot*) for each contributor from the GitHub REST API's `users` endpoint on 9 November 2021. Fig. 1.5 ranks the top 20 contributors to these 10 popular software projects in decreasing order of contributed commits. Contributors that are responsible for at least 1% of all commits are highlighted. We classified the contributors into three categories (see Fig. 1.4): human account, bot actor and bot account. Contributors belonging to the third category are confirmed to be machine users through a manual inspection of their activities by two co-authors of Golzadeh et al. (2022b).

Fig. 1.5 shows that two among the considered 10 repositories use only bot actors, while eight repositories have between one and three bot accounts in their top 20 contributors. Also, more than half of these bots (12 out of 21) are bot accounts (machine users). If we focus on the subset of contributors that made at least 1% of all commits, the overwhelming majority of bot actors (8 out of 9) and bot accounts (10 out of 12) belong to those contributors. On average, these bots are responsible for nearly one fifth of all commits in these projects.

In summary, this preliminary evidence highlights that bots carry out a significant amount of work and many of them belong to the most active committers to the repository. Nevertheless, none of the bot accounts were identified as such by GitHub.

1.5. Thesis statement and goals

The previous section focussed only on bots that are involved in committing. It is very likely that they are involved in many other activity types too given that collaborative software development on GitHub involves a wide range of activity types such as opening issues, creating PRs, publishing releases, performing code reviews and so on (Dabbish et al., 2012). As some activities can be effort-intensive, repetitive and error-prone, they are likely to be automated using some or all of the automation mechanisms presented in Section 1.2. The usage of bots in collaborative software development helps to reduce developer's workload, allowing them to focus on more complex and mentally challenging tasks (Wessel et al., 2018).

Detecting the presence of bots in GitHub repositories is important for software engineering researchers performing socio-technical studies and productivity analyses of software development projects in GitHub. Disregarding the contributions of bots or not treating their activities differently from humans could lead to biased, incorrect and misleading conclusions during empirical analyses (Cassee et al., 2021; Dey et al., 2020a,b; Zhang et al., 2022). Additionally, it is essential to correctly recognise the contributions made by human contributors since their activities in collaborative software development are used as a criterion for employers when hiring developers (Hauff & Gousios, 2015), providing accredition (Hann et al., 2002), and to understand and improve project development (Liao et al., 2020). As witnessed by initiatives such as the CHAOSS Linux Foundation project[15] and associated software development analytics tools such as GrimoireLab,[16] it is important to assess the health of a software community (Oriol et al., 2023) by considering all activities of each contributor.

Several bot identification approaches have been proposed previously (Dey et al., 2020a; Golzadeh et al., 2021b, 2020; Abdellatif et al., 2022) as it can be challenging to distinguish bots from human contributors. However, a large amount of data of different nature is required by some of these approaches to identify bots, making them difficult to use at scale. Furthermore, most of these approaches detect bots by

[15]`https://chaoss.community`
[16]`https://chaoss.github.io/grimoirelab/`

considering features based on a limited set of activity types, e.g., only commit related activities Dey et al. (2020a) or only issue and PR related activities (Golzadeh et al., 2021b). These observations led me to formulate the following thesis statement:

Thesis statement:
It is important to identify bots in GitHub repositories. Existing bot identification approaches suffer from various limitations. We propose improved bot identification approaches that overcome these limitations.

The following goals are proposed to address the thesis statement.

Goal 1: *Improve existing bot identification approaches to overcome their limitations.*

Some bot identification approaches work at the level of individual GitHub repositories. This might give rise to different predictions in different GitHub repositories for the same contributor. An enhanced bot identification model could combine the predictions made for a contributor on multiple repositories and provide a single prediction of the contributor type.

Goal 1 will provide two improved bot identification models. The first model leverages the predictions made by the approach of Golzadeh et al. (2021b) in all considered GitHub repositories to determine the contributor type. The second model is an ensemble model that combines the predictions made by multiple bot identification approaches.

Goal 2: *Develop a new bot identification model that leverages a wide range of activity types performed by contributors.*

In GitHub, bots and human contributors perform many activity types. Existing bot identification approaches depend on features based on a limited set of activity types to detect bots in GitHub. This makes these approaches unable to detect a bot that is involved only in specific activity types that are not considered by them. This highlights the usefulness of an improved model that can identify bots in GitHub based on a wider range of contributor activity types.

Goal 2 will have three outcomes. A curated dataset of manually labelled bots and human contributors and their associated activity sequences in GitHub. A set of behavioural features based on their activity sequences that can statistically differentiate bots from human contributors. A classification model to identify bots based on these features.

Goal 3: *Leverage the practical use of the activity-based bot identification model.*

There has been little research on how GitHub's automation mechanisms are used in the context of large software ecosystems composed of a large community of collaborating contributors. Conducting such studies helps to understand the roles and dynamics of bots in large software ecosystems.

Goal 3 will use the activity-based bot identification model resulting from Goal 2, and will have two contributions. First, it will reveal the differences in activity patterns between bots and humans in a large software ecosystem, and also between GitHub internal automation services, GitHub Apps and bot accounts. Second, we present an efficient command-line based open-source bot identification tool that can be used at scale to determine the contributor type of thousands of contributors per hour.

1.6. Thesis structure

This dissertation is comprised of eight chapters. The current chapter covered the research context, thesis statement and thesis goals and contributions. Section 1.4 was based on the following research publication that I co-authored.

- Golzadeh, M., Mens, T., Decan, A., Constantinou, E., & Chidambaram, N. (2022b). Recognizing bot activity in collaborative software development. IEEE Software, 39 (5), 56–61. DOI: 10.1109/MS.2022.3178601 (Golzadeh et al., 2022b).

The remainder of this dissertation is structured as follows. *Chapter 2* provides the background required to understand the concepts explored in this thesis. This chapter provides an overview of GitHub's automation practices and APIs, and illustrates their use with some concrete examples.

Chapter 3 presents the related work. It provides an overview of research publications that report on empirical studies on bots in GitHub. The chapter discusses on different categories and usages of bots, challenges with using bots, developers' and maintainers' perspectives on using bots, guidelines for developing bots, and so on. The chapter continues by presenting existing bot identification approaches in GitHub and concludes with an overview of how these approaches have been used to identify bots in various research studies.

Chapters 4 to *7* are based on peer-reviewed scientific articles. Each article has at least one outcome in the form of a dataset, model and so on. Fig. 1.6 visually presents for each chapter the article(s) covered in it, the contributions and the goal addressed by them. *Chapter 4* addresses **Goal 1**. It proposes two improved bot identification models. First, it presents an ensemble classification model that combines the prediction results provided by multiple bot identification approaches to identify bots in GitHub. This model identifies bots by considering multiple activity types rather than depending on a specific activity type. Second, the chapter presents another bot identification model that improves the predictions made by BoDeGHa (Golzadeh et al., 2021b), an existing bot identification model. For contributors active in multiple GitHub repositories, the improved model combines the predictions made for them in each of these repositories separately to give its final prediction on the contributor type (*bot* or *human*). The improved models presented in this chapter provide a better precision and recall than the original approaches. The analyses and results presented in this chapter are based on the following peer-reviewed publications:

- Golzadeh, M., Decan, A., & Chidambaram, N. (2022). On the accuracy of bot detection techniques. In International Workshop on Bots in Software Engineering (BotSE). IEEE. DOI: 10.1145/3528228.3528406 (Golzadeh et al., 2022a).

- Chidambaram, N., Decan, A., & Golzadeh, M. (2022). Leveraging predictions from multiple repositories to improve bot detection. In International Workshop on Bots in Software Engineering (BotSE). IEEE. DOI: 10.1145/3528228.3528403 (Chidambaram et al., 2022).

Chapters 5 and *6* contribute to **Goal 2** of this thesis. *Chapter 5* provides a dataset of bot and human activities in GitHub and introduces behavioural features that can be used to differentiate bots from human contributors. First it presents a curated ground-truth dataset of bot and human contributors. Then, it presents an automated process to convert low-level contributor events to high-level activities. Based on this process, it introduces a dataset of contributor activity sequences. Using this dataset, this chapter statistically evaluates five behavioural features that can be used to differentiate bots from humans in GitHub. The results reported in this chapter are based on the following peer-reviewed publications:

- Chidambaram, N., Decan, A., & Mens, T. (2023). A dataset of bot and human activities in GitHub. In International Conference on Mining Software Repositories (MSR) Data and Tool Showcase Track, pp. 465–469. IEEE/ACM. DOI: 10.1109/MSR59073.2023.00070 (Chidambaram et al., 2023a).

- Chidambaram, N., Decan, A., & Mens, T. (2023). Distinguishing bots from human developers based on their GitHub activity types. In Seminar on Advanced Techniques & Tools for Software Evolution (SATToSE), Vol. 3483, pp. 31-39. CEUR Workshop Proceedings. Publication: `https://ceur-ws.org/Vol-3483/paper3.pdf` (Chidambaram et al., 2023b).

Chapter 6 completes **Goal 2** by providing a manually labelled ground-truth dataset of bot and human contributors in GitHub, and using this dataset to train a machine learning-based model that can identify bots in GitHub. First, the chapter extends the ground-truth contributor dataset of *Chapter 5* with new manually labelled contributors. Second, it identifies and quantifies limitations of existing bot identification approaches in terms of precision, recall, volume of data downloaded, execution time and number of required API queries. Third, it explores a wider range of features that can differentiate bots from human contributors. Finally, it details the procedure followed to train and evaluate BIMBAS, a binary classification model to distinguish bots from human contributors in GitHub. The contributions presented in this chapter are based on the following peer-reviewed journal publication:

- Chidambaram, N., Decan, A., & Mens, T. (2024). A Bot Identification Model and Tool Based on GitHub Activity Sequences. In Journal of Systems and Software (JSS), Elsevier, vol. 221, ISSN: 0164-1212. DOI: 10.1016/j.jss.2024.112287 (Chidambaram et al., 2025).

Chapter 7 addresses **Goal 3**. First, it presents the usage of BIMBAS to perform large-scale empirical studies in a large software ecosystem that has at least hundreds of contributors and repositories. On the one hand we identify the differences in activity patterns between bots and humans and on the other hand between different categories of bots. Second, based on the BIMBAS classification model, this chapter presents and evaluates the efficiency of RABBIT, a command-line bot identification tool that can be used at scale to determine the contributor type of thousands of contributors per hour. The results presented in this chapter are based on the following peer-reviewed publications:

- Chidambaram, N., & Mens, T. (2025). Observing bots in the wild: A quantitative analysis of a large open source ecosystem. In International Workshop on Bots in Software Engineering (BotSE), IEEE (Chidambaram & Mens, 2025).
- Chidambaram, N., Decan, A., & Mens, T. (2024). A Bot Identification Model and Tool Based on GitHub Activity Sequences. In Journal of Systems and Software (JSS), Elsevier, vol. 221, ISSN: 0164-1212. DOI: 10.1016/j.jss.2024.112287 (Chidambaram et al., 2025).
- Chidambaram, N., Mens, T., & Decan, A. (2024). RABBIT: A tool for identifying bot accounts based on their recent GitHub event history. In International Conference on Mining Software Repositories (MSR) Data and Tool Showcase Track. ACM. DOI: 10.1145/3643991.3644877 (Chidambaram et al., 2024).

Chapter 8 summarises all achieved research contributions and how they contributed towards the thesis statement. The chapter also presents the limitations and discussions of this dissertation. Finally, it discusses on the perspectives that can keep this research going on.

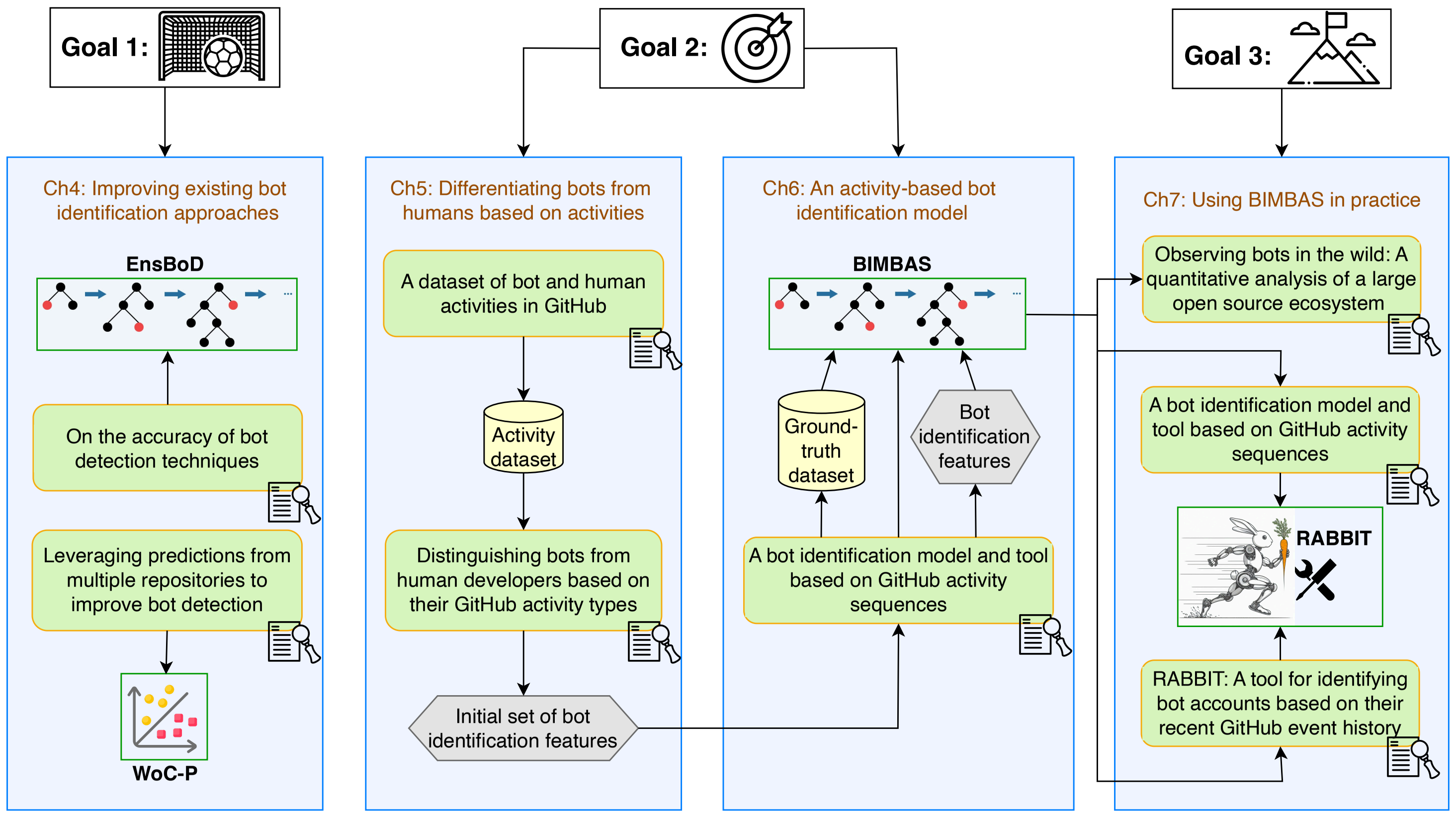

Goal 1:
Goal 2:
Goal 3:
Ch4: Improving existing bot identification approaches
EnsBoD
On the accuracy of bot detection techniques
Leveraging predictions from multiple repositories to improve bot detection
WoC-P
Ch5: Differentiating bots from humans based on activities
A dataset of bot and human activities in GitHub
Activity dataset
Distinguishing bots from human developers based on their GitHub activity types
Initial set of bot identification features
Ch6: An activity-based bot identification model
BIMBAS
Ground-truth dataset
Bot identification features
A bot identification model and tool based on GitHub activity sequences
Ch7: Using BIMBAS in practice
Observing bots in the wild: A quantitative analysis of a large open source ecosystem
A bot identification model and tool based on GitHub activity sequences
RABBIT
RABBIT: A tool for identifying bot accounts based on their recent GitHub event history

Figure 1.6: Structure, goals and research outcomes of this dissertation.

CHAPTER 2

Background

In collaborative software projects, developers and repository maintainers tend to use various services for automating repetitive, effort-intensive and error-prone tasks. Automation services in GitHub perform their activities under the name of their bot actors or bot accounts. As mentioned in Section 1.3, we collectively refer to them as bots. Since activities performed by bots originate from different automation services, it is required to understand the usage of these services in GitHub.

This chapter explains various automation mechanisms available in GitHub, that are often used together in repositories. First, we discuss GitHub Actions workflows, which are configurable automated processes used to perform tasks like building, testing and deploying software. Second, we discuss GitHub's internal automation services, which are readily available tools provided by GitHub for all its repositories. Third, we discuss GitHub Apps, which are tools that can be installed in GitHub organisations and repositories to automate tasks. Fourth, we discuss bot accounts, which are personal user accounts handled by machine users for automating various tasks. Next, we present GitHub's external automation services, which are mechanisms that can send information to external systems to perform some tasks. Finally, we explain the GitHub APIs that allow developers, maintainers and automation services to interact with GitHub, retrieve or modify GitHub related data, and manage and perform activities in repositories.

2.1. GitHub automation mechanisms

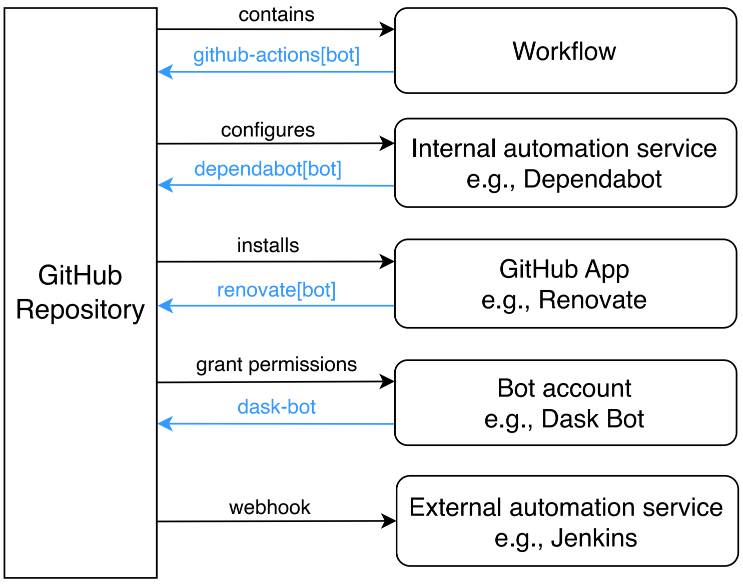

GitHub Repository
contains
github-actions[bot]
Workflow
configures
dependabot[bot]
Internal automation service e.g., Dependabot
installs
renovate[bot]
GitHub App e.g., Renovate
grant permissions
dask-bot
Bot account e.g., Dask Bot
webhook
External automation service e.g., Jenkins

Figure 2.1: Automation mechanisms in GitHub.

Fig. 2.1 shows the automation mechanisms that are available in GitHub.

A GitHub repository can contain *workflows*. They are frequently used for Continuous Integration (CI) and Continuous Deployment (CD) tasks, but can automate many other tasks as well. For example, a workflow can be configured to post a comment in issue/PR or create/delete labels. It can be triggered by a commit, a PR, at scheduled time and so on. Its activities are by default visible through GitHub UI under the name of specific bot actor called github-actions[bot].

GitHub Apps are another automation mechanism. They can be installed in a GitHub repository and configured to perform various tasks such as merging PRs, opening issues, performing code reviews and so on. Apps can be installed in repositories or in an organisation account to perform its tasks. Similar to workflows, activities performed by Apps are visible through the GitHub UI under the name of their corresponding bot actors. For example, activities performed by `Renovate`[1] App will look like they are performed by its bot actor renovate[bot] in GitHub repositories.

GitHub also provides *internal automation services* that can be activated on repositories. For example, `Dependabot`[2] allows to check for dependency and security updates, and the `CodeQL`[3] analysis service allows to check for vulnerabilities and code

[1]`https://github.com/apps/renovate`
[2]`https://github.com/dependabot`
[3]`https://codeql.github.com/`

errors. The internal automation mechanisms can be configured in a GitHub repository through the repository settings. The activities performed by the internal automation mechanisms are visible through the GitHub API under the name of their respective bot actors, for example dependabot[bot] in case of `Dependabot` and github-actions[bot] in the case of `CodeQL`.

Activities in GitHub can also be automated through *bot accounts* which are present in GitHub as personal user accounts. They require permission from a repository maintainer to collaborate in that GitHub repository to perform its intended activities. Their activities are visible in through the GitHub UI under their own personal user account name. For example, activities performed by `Dask Bot`,[4] a bot for automating Dask organisation management tasks, will be visible in the GitHub API under its login name dask-bot.

Finally, there are the *external automation services*, which provide notifications to external systems through webhooks configured in the GitHub repository. Also, they can trigger other automation mechanisms in response to specific activity types. For example, `Jenkins`[5] provides plugins to support building, deploying and automating any project. It allows users to set up webhooks that trigger `Jenkins` every time a change is pushed to GitHub. Similarly, `SonarQube Cloud`, a cloud-based static code analysis service that performs continuous code quality and security checks, provides a GitHub App[6] that can be installed in GitHub repositories or organisations. External automation services usually communicate back to GitHub through any of the other automation mechanisms.

The next sections in this chapter provide details about each automation mechanism in GitHub.

2.2. GitHub Actions workflows

Continuous integration, deployment and delivery (CI/CD) have become widespread in collaborative software development (Fairbanks et al., 2023). GitHub introduced the GitHub Actions in October 2018 to support CI/CD for its repositories, allowing developers to automate their workflows directly within GitHub without the need for any external CI/CD automation service like Travis[7] or Jenkins. It enables to run workflows for automating a wide variety of activities in a repository such as code reviewing, verifying licence agreements, monitoring and updating dependencies and fixing security vulnerabilities.

A GitHub Actions workflow is a configurable automated process to execute one or more jobs (e.g., to build, test and deploy software) in a repository. The workflows that need to be executed are written in YAML language and can be found as a .yml file in the .github/workflows folder of the GitHub repository. A repository can contain multiple workflow files that can perform different tasks. Each workflow has multiple

[4]`https://github.com/dask-bot`
[5]`https://www.jenkins.io/solutions/github/`
[6]`https://github.com/marketplace/sonarcloud`
[7]`https://www.travis-ci.com/`

components such as trigger, job, runner and step.

An example of a workflow file[8] is given in Listing 2.1. This workflow is triggered when a PR is created in the repository and posts the comment 'Thanks for reporting!' under the PR. We will use this example to introduce the syntactic elements used in a workflow.

Listing 2.1: Example of a workflow file.

```
1  on:
2      pull_request:
3
4  jobs:
5      comment:
6          runs—on: ubuntu—latest
7          steps:
8              — uses: actions/github—script@60a0d83039c74a4aee543508d2ffcb1c3799cdea
9                with:
10                 script: |
11                     github.rest.issues.createComment({
12                         issue_number: context.issue.number,
13                         owner: context.repo.owner,
14                         repo: context.repo.repo,
15                         body: 'Thanks for reporting!'
16                    })
```

Trigger: A workflow can be triggered to run based on various events. For example, the workflow in Listing 2.1 gets triggered when the event specified on line 2 happens in a repository i.e., opening a PR. Other possible triggers for a workflow can be a release event (publishing a release), an issue event (opening an issue), a push event (committing files), a scheduled time (e.g., at 6:00AM on every Monday) and so on.

Job: Once a workflow is triggered, it executes the jobs specified in it (line 5). A job (line 4) defines a set of one or more steps (line 7) that need to be sequentially executed.

Runner: A runner is a machine with pre-installed software and tools that can be used to execute a workflow. For example, line 6 specifies that the workflow needs to be executed on the latest version of ubuntu OS. A workflow can also be executed on other GitHub-hosted runners (e.g., windows-latest and macos-latest).

Step: A step is the smallest unit of work in a workflow. It can be an Action that can be executed in a workflow. For example, line 8 specifies the repository hosting the Action (actions/github-script) along with the version to be used (@60a0d...). The workflow first checks out the repository containing the Action (line 8), and then executes the script (lines 11-15) that posts a comment 'Thanks for reporting' under the PR that was created.

Action: An Action is a reusable automation component that can perform a particular task (e.g., actions/github-script on line 8). The GitHub Marketplace[9] provides

[8] `https://github.com/tommens/calculator-cucumber/blob/master/.github/workflows/comment.yml`
[9] `https://github.com/marketplace?type=actions`

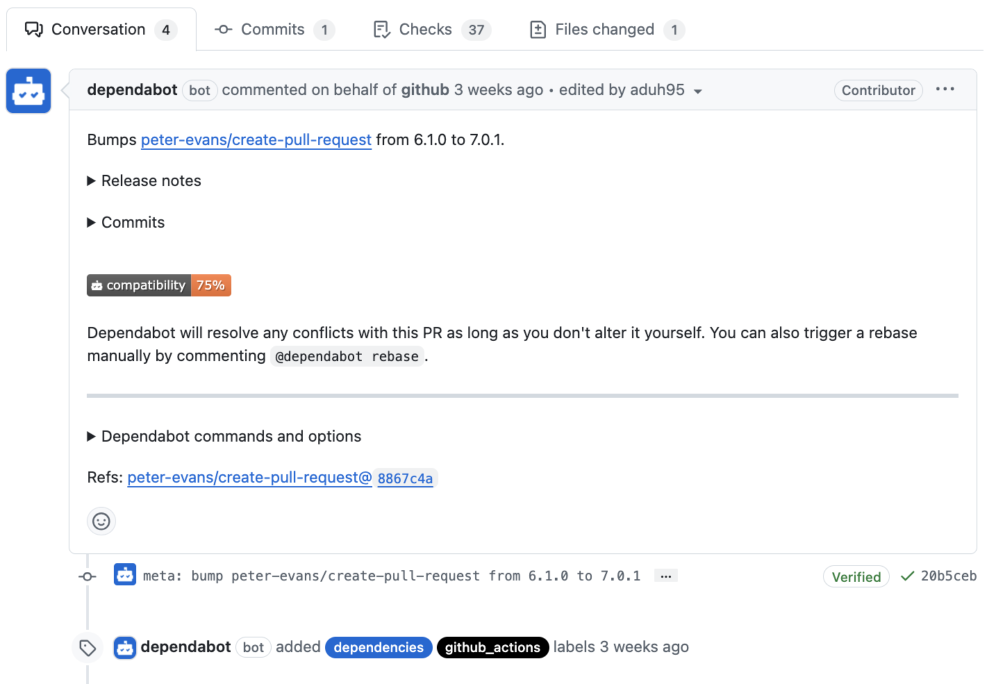
many pre-written Actions (Wessel et al., 2023a) that can be used in a workflow to avoid explicitly writing the commands that are commonly used.

By default, the activities carried out by GitHub Actions workflows are visible in the GitHub UI under the name of a specific bot actor called github-actions[bot]. For example, an activity done by the workflow provided in Listing 2.1 is given in Fig. 1.2 (in Section 1.3 on page 6), where github-actions[bot] posts the comment 'Thanks for reporting' as soon as the PR is created.

2.3. Internal automation services

GitHub provides a set of automation services that are available for all its repositories.

`Dependabot` is an internal automation service for keeping project dependencies up-to-date, providing security updates, and alerting about vulnerabilities that affect dependencies. Fig. 2.2 gives an example of a PR created by `Dependabot` in *nodejs/node* repository. This PR is created to update one of the project dependencies from version 6.1.0 to version 7.0.1. It provides a compatibility score and adds labels too.



dependabot bot commented on behalf of github 3 weeks ago • edited by aduh95 Contributor

Bumps peter-evans/create-pull-request from 6.1.0 to 7.0.1.

▶ Release notes

▶ Commits

compatibility 75%

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.

▶ Dependabot commands and options

Refs: peter-evans/create-pull-request@ `8867c4a`

`meta: bump peter-evans/create-pull-request from 6.1.0 to 7.0.1` … Verified ✓ `20b5ceb`

dependabot bot added dependencies github_actions labels 3 weeks ago

Figure 2.2: A PR created in *nodejs/node* repository by `Dependabot` under the name of its bot actor dependabot[bot].

`CodeQL` is another internal automation service of GitHub. It identifies vulnerabilities and errors in the code. The results will be shown as alerts in the *Code scanning* tab under the security settings of the repository. The service can be configured in three ways. First, by using its default configuration that automatically identifies the language, query suite to run and events that trigger scans. Second, by using a customised version of `CodeQL` analysis by creating and editing a workflow file or configuring third-party Actions. Third, by using the `CodeQL` CLI or an external CI system and upload the results to GitHub. The service reports on the code scanning severity such as Error, Warning or Note, and on security severity such as Critical, High, Medium or Low.

`Merge Queue` is an internal automation service provided by GitHub to control branch traffic, and automate PR merges into a busy branch and ensuring that the branch is never broken by incompatible changes. When a PR is added to the merge queue, it creates a temporary branch with the latest version of the base branch, merges the PRs that are already in queue before merging the latest PR that is added to the queue and executes CI services. Upon successful CI results, the new PR is added to the merge queue. Through `Merge Queue`, a working system of the project is ensured at every stage of development.

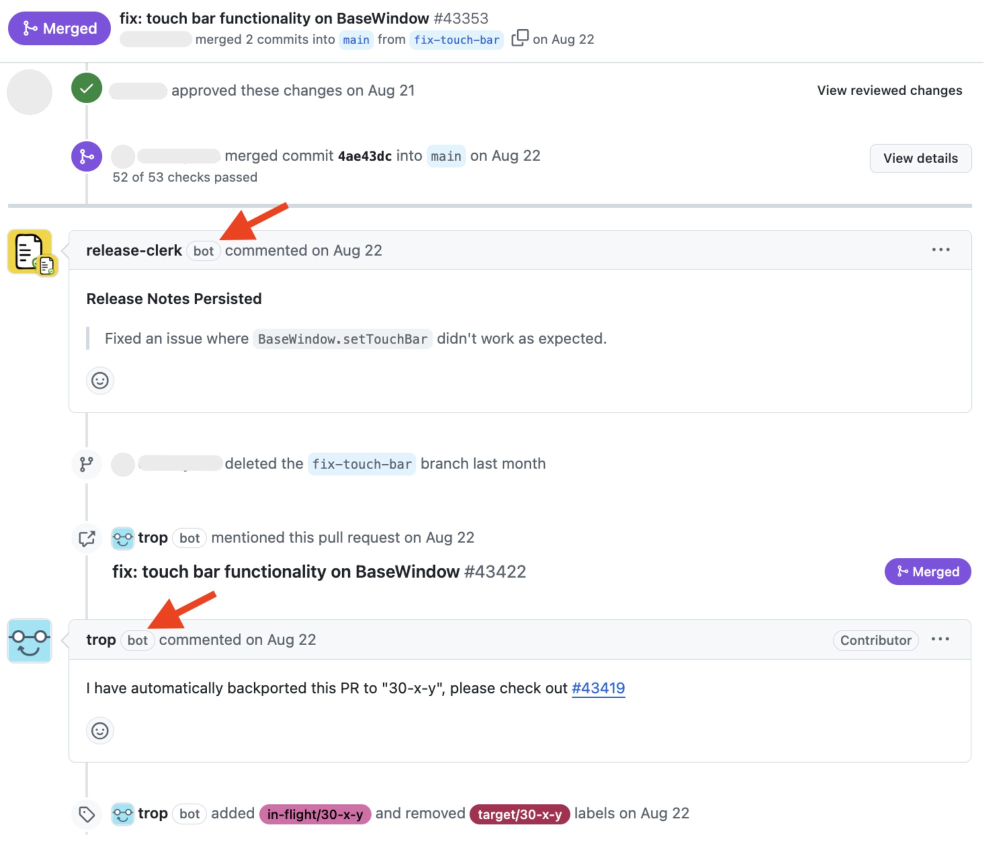
Activities performed by `Merge Queue` is visible in GitHub UI under the name of its bot actor github-merge-queue[bot].

2.4. GitHub Apps

GitHub Apps are tools that extend GitHub's functionality. They can be installed on a user account or organisation and used in the GitHub repositories owned by that account or organisation to perform various activities such as opening issues and PRs, commenting on issues and PRs, creating or deleting tags and branches and so on. Based on the events that happen in repositories, GitHub App can be triggered to perform activities outside GitHub. For example, a GitHub App can post on Slack when an issue is opened in the corresponding GitHub repository.[10]

GitHub Apps can be triggered in various ways such as manually, by workflows and by specific events that are performed in repositories. For example Fig. 2.3 shows an example of a PR in the *electron/electron* GitHub repository where `Release Clerk`[11] App is triggered once the PR is merged. `Release Clerk` is a publicly available third-party App (not distributed through GitHub Marketplace) that verifies if PRs have release notes. Once the App finished its tasks, it posts a comment in the corresponding PR which will appear through the GitHub UI as an activity performed by its bot actor release-clerk[bot] (as indicated by the first red arrow in Fig. 2.3). Next, `trop` is triggered, a private GitHub App managed by the Electron organisation for backporting PRs (applying changes to a stable branch from the current branch), and adding and removing tags in PRs. The activities done by `trop` GitHub App are visible through the GitHub UI under the name of its bot actor trop[bot] (as indicated by the second red arrow in Fig. 2.3).

[10]`https://github.com/integrations/slack`
[11]`https://github.com/apps/release-clerk`



Merged **fix: touch bar functionality on BaseWindow** #43353
[redacted] merged 2 commits into `main` from `fix-touch-bar` on Aug 22

[redacted] approved these changes on Aug 21 — View reviewed changes

[redacted] merged commit **4ae43dc** into `main` on Aug 22 — View details
52 of 53 checks passed

release-clerk bot commented on Aug 22

Release Notes Persisted

> Fixed an issue where `BaseWindow.setTouchBar` didn't work as expected.

[redacted] deleted the `fix-touch-bar` branch last month

trop bot mentioned this pull request on Aug 22
fix: touch bar functionality on BaseWindow #43422 — Merged

trop bot commented on Aug 22 — Contributor

I have automatically backported this PR to "30-x-y", please check out #43419

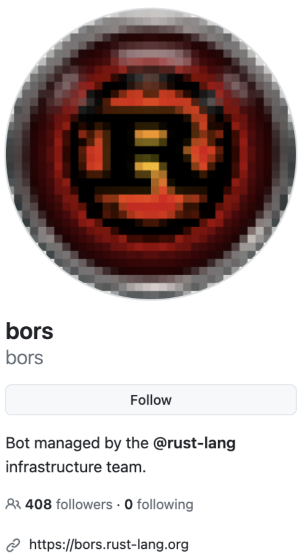
trop bot added in-flight/30-x-y and removed target/30-x-y labels on Aug 22

Figure 2.3: Example of GitHub App commenting in PR.

2.5. Bot accounts

To access a GitHub repository, one should own a GitHub account, which can be of three types. A *personal account* is required for everyone using GitHub. These accounts can own an unlimited number of public and private repositories with an unlimited number of collaborators and the activities that they perform on GitHub will be attributed to their GitHub account. An *organisation account* serves as a container for team members to collaborate on GitHub. Multiple contributors, each having their own personal account, can collaborate on shared projects by joining the same organisation account. A subset of these personal accounts can be given the role of organisation owner, which allows those people to granularly manage access to the organisation's resources using sophisticated security and administrative features. An *enterprise account* allows administrators to centrally manage policy and billing for multiple organisations and enable innersourcing between the organisations.

Some GitHub personal accounts can be configured to be used by an automated



bors
bors

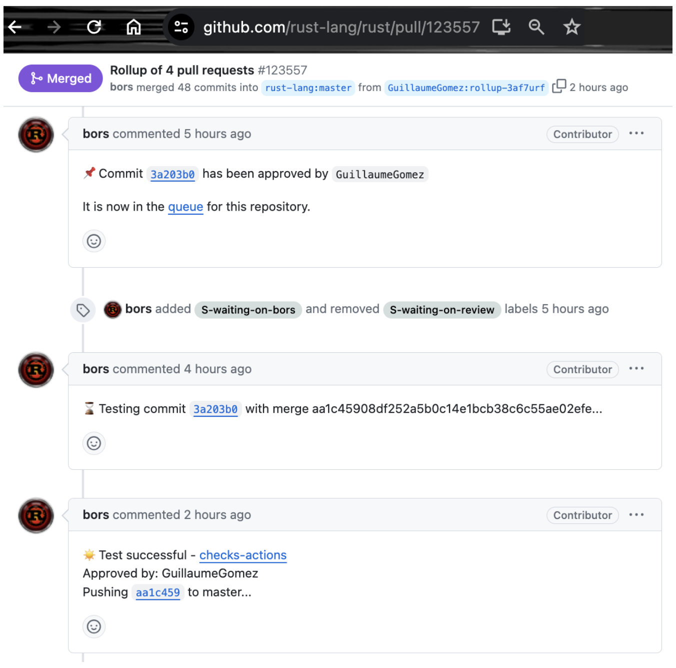
Follow

Bot managed by the @rust-lang infrastructure team.

408 followers · 0 following

https://bors.rust-lang.org

(a) Bors - bot account



github.com/rust-lang/rust/pull/123557

Merged **Rollup of 4 pull requests** #123557
bors merged 48 commits into `rust-lang:master` from `GuillaumeGomez:rollup-3af7urf` 2 hours ago

bors commented 5 hours ago — Contributor

📌 Commit `3a203b0` has been approved by `GuillaumeGomez`

It is now in the queue for this repository.

bors added `S-waiting-on-bors` and removed `S-waiting-on-review` labels 5 hours ago

bors commented 4 hours ago — Contributor

⌛ Testing commit `3a203b0` with merge aa1c45908df252a5b0c14e1bcb38c6c55ae02efe...

bors commented 2 hours ago — Contributor

☀️ Test successful - checks-actions
Approved by: GuillaumeGomez
Pushing `aa1c459` to master...

(b) Bors activity in a PR

Figure 2.4: Bors GitHub personal account profile and an example of its activity in a PR in *rust-lang/rust* repository.

actor to automate certain intended tasks in GitHub repositories. Such accounts are called bot accounts. They can be regarded as machine users of artificial software developers (Erlenhov et al., 2019) since they can perform all the activities that a human can perform in GitHub through its personal account. Bot accounts are used for automating a wide variety of tasks (Wessel et al., 2018) such as responding to events (e.g., PR) and managing the flow of a project. Bot accounts can also interact with, react to, or trigger other automation mechanisms such as workflows, Apps, internal and external automation services.

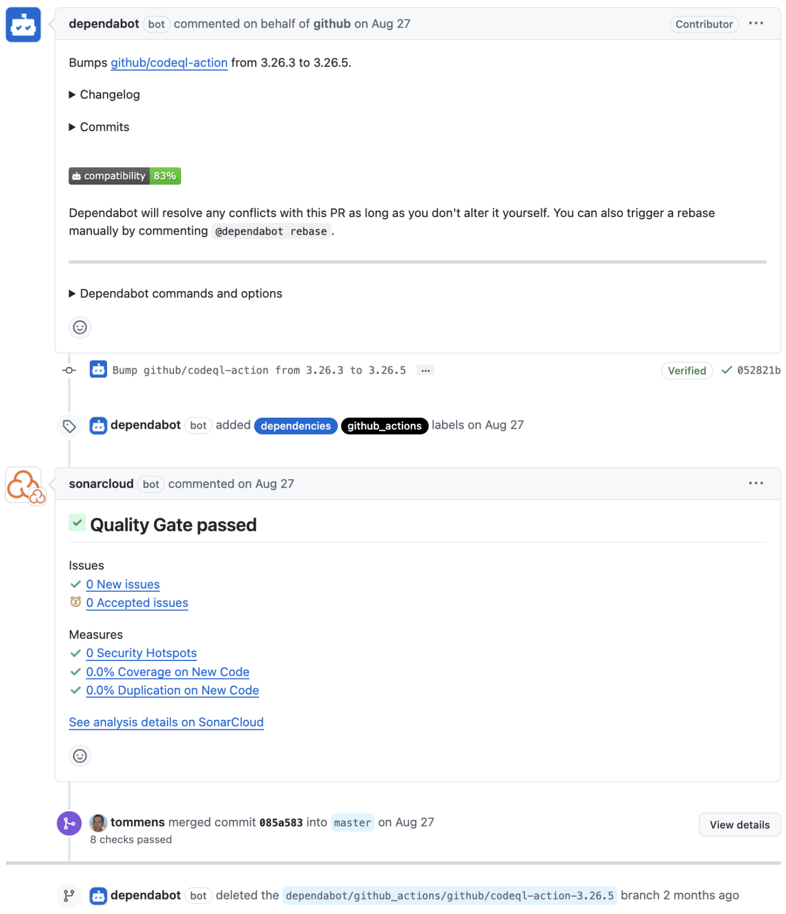
Fig. 2.4a shows the profile of bors, a bot account in GitHub that is maintained by the Rust language[12] team. Rust is a systems programming language created by the Mozilla team. The *Rust-lang* organisation in GitHub uses bors for various tasks such as commenting on PRs, adding/removing labels of a PR, merging commits from PRs, closing a PR and deleting its corresponding branch, closing issues associated with PRs and so on. Fig. 2.4b gives an example of three successive tasks performed by bors: It first merges PR commits to the master branch, then comments on the PR, and finally updates the PR label.

Unlike Apps, GitHub does not recognise and label bot accounts as 'bot'. So, a

[12]`https://github.com/rust-lang`

user may not necessarily realise that a task is automated by a bot account. This highlights the need for GitHub bot identification tools.

2.6. External automation services



dependabot bot commented on behalf of github on Aug 27 Contributor ···

Bumps github/codeql-action from 3.26.3 to 3.26.5.

▶ Changelog

▶ Commits

compatibility 83%

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.

▶ Dependabot commands and options

`Bump github/codeql-action from 3.26.3 to 3.26.5` ··· Verified ✓ `052821b`

dependabot bot added dependencies github_actions labels on Aug 27

sonarcloud bot commented on Aug 27 ···

✓ **Quality Gate passed**

Issues
✓ 0 New issues
0 Accepted issues

Measures
✓ 0 Security Hotspots
✓ 0.0% Coverage on New Code
✓ 0.0% Duplication on New Code

See analysis details on SonarCloud

tommens merged commit `085a583` into `master` on Aug 27 View details
8 checks passed

dependabot bot deleted the `dependabot/github_actions/github/codeql-action-3.26.5` branch 2 months ago

Figure 2.5: A PR created by dependabot[bot] in repository *tommens/calculator-cucumber* where the external system `https://sonarcloud.io` performs code quality analysis and posts the report as comment using its sonarcloud[bot] bot actor corresponding to `SonarCloud` GitHub App.

GitHub provides access to external automation services through webhooks. These webhooks allow to notify external systems whenever certain events occur on GitHub. A webhook can be created within a specific repository, organisation, or GitHub App. The resources that a webhook can access depend on where it is installed. For example, Fig. 2.5 shows an automatic PR created by Dependabot to update a dependency. The creation of this PR triggers the execution of a GitHub App SonarQube Cloud that checks for several code quality issues in the code, resulting in a PR comment with the code quality report posted by its bot actor sonarcloud[bot]. It is the SonarQube Cloud GitHub App that accesses the external cloud service `https://sonarcloud.io` that performs this code quality analysis. After the quality checks are passed, a human user tommens accepts and merges this PR and Dependabot deletes the branch it created for this PR.

Based on some events performed in a repository such as a PR being opened or an issue being created, webhooks can trigger GitHub automated workflows (Chandrasekara & Herath, 2021). Also, they can be used to send notification such as to Slack whenever an event (e.g., new commit, comment and a PR is created) is performed in GitHub, or updating some external issue tracker (e.g., Jira).

2.7. GitHub REST API

API is an acronym for Application Programming Interface. It provides a connection between computers, computer programs or pieces of software to interact with each other. GitHub offers many APIs, but in this thesis, we rely only on its REST API.[13] It allows users and automation services to interact with GitHub artifacts (pertaining to repositories, organisations, users and so on) such as issues, PRs, tags, branches and so on. Further, it can be used to automate various activities such as creating or deleting repositories, creating or deleting tags, creating or deleting branches, committing files, publishing releases and so on. Interaction with REST API can be made using GitHub's CLI, curl (a command line tool for transferring data), the Octokit official clients for the GitHub API, and other third-party libraries. It provides data in a pre-determined structure. This section provides details about the GitHubREST API.

The GitHub REST API has many targeted endpoints that can be used to send or receive information regarding functionalities such as users (get information about personal accounts, Apps and internal automation services), issues (interact with issues), pulls (interact with PRs), repositories (interact with repositories), events (get public events performed by a user, public events performed in a repository and public events performed in an organisation) and so on.[13]

For example, to get the first 30 followers of a GitHub user account, the following GitHub REST API query to its `users` endpoint needs to be made: `https://api.github.com/users/<username>/followers`. Listing 2.2 shows the information retrieved through this query regarding the fist 30 followers of natarajan-chidambaram.[14]

If one would like to obtain the first three followers of the first two followers of natarajan-chidambaram, first a script needs to be written to parse the JSON result of Listing 2.2 and to extract the value stored in the "login" key for the first two followers. Then, two more queries need to be made: `https://api.github.com/users/human1/`

[13] `https://docs.github.com/en/rest`

[14] Usernames are pseudo-anonymised to comply to GDPR guidelines.

`followers` and `https://api.github.com/users/human2/followers`. Overall, it requires three API queries and some additional programming to obtain the required information about the followers.

Listing 2.2: Output provided by the REST API for the query to fetch the first 30 followers of natarajan-chidambaram's GitHub account.

```
[
   {
     "login": "human1",
     "id": 12345678,
     "node_id": "ABcd123=",
     "url": "https://api.github.com/users/human1",
     "html_url": "https://github.com/human1",
     "followers_url": "https://api.github.com/users/human1/followers",
     .
     .
     .
     "type": "User",
     "site_admin": false
   },
   {
     "login": "human2",
     "id": 23456789,
     "node_id": "PQRst123=",
     "url": "https://api.github.com/users/human2",
     "html_url": "https://github.com/human2",
     "followers_url": "https://api.github.com/users/human2/followers",
     .
     .
     .
     "type": "User",
     "site_admin": false
   },
   .
   .
   .
]
```

Listing 2.3: Using the GitHub REST API to create a new issue in a repository.

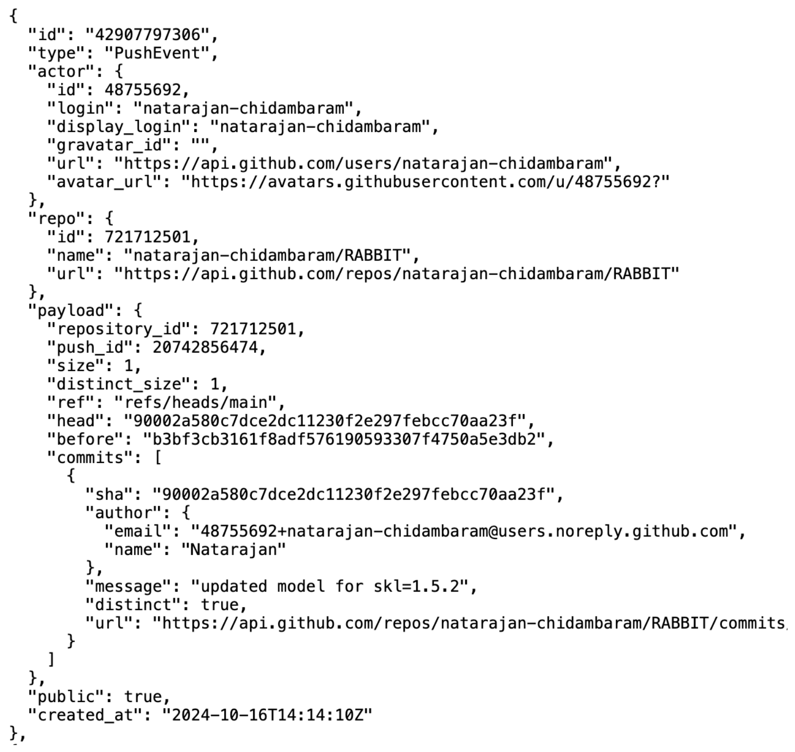
```
1 curl -X POST \
2   -H "Authorization: Bearer <token>" \
3   -H "Accept: application/vnd.github+json" \
4   https://api.github.com/repos/<repo_owner>/<repo_name>/issues \
5   -d '{"title": "title", "body": "description", "labels": ["l1"]}'
```

Apart from retrieving data, the REST API can also be used to post or update data on GitHub. For example, one can manage issues, PRs, assignees, labels, branches, discussions, and tags. Listing 2.3 provides an example for creating an issue in a repository using the GitHub REST API through a terminal.[15] Line 1 conveys that an HTTP POST method is used to send data to GitHub. In line 2, the user has to enter their access token, and in line 4, the repository owner and repository name has

[15] `https://docs.github.com/en/rest/issues/issues?#create-an-issue`

to be entered. Line 5 is the data (in the format of key-value pair) corresponding to the issue that is to be created, where "title" (corresponds to the title of the issue) is the only mandatory parameter.

`Events` endpoint of the GitHub REST API:



```
{
  "id": "42907797306",
  "type": "PushEvent",
  "actor": {
    "id": 48755692,
    "login": "natarajan-chidambaram",
    "display_login": "natarajan-chidambaram",
    "gravatar_id": "",
    "url": "https://api.github.com/users/natarajan-chidambaram",
    "avatar_url": "https://avatars.githubusercontent.com/u/48755692?"
  },
  "repo": {
    "id": 721712501,
    "name": "natarajan-chidambaram/RABBIT",
    "url": "https://api.github.com/repos/natarajan-chidambaram/RABBIT"
  },
  "payload": {
    "repository_id": 721712501,
    "push_id": 20742856474,
    "size": 1,
    "distinct_size": 1,
    "ref": "refs/heads/main",
    "head": "90002a580c7dce2dc11230f2e297febcc70aa23f",
    "before": "b3bf3cb3161f8adf576190593307f4750a5e3db2",
    "commits": [
      {
        "sha": "90002a580c7dce2dc11230f2e297febcc70aa23f",
        "author": {
          "email": "48755692+natarajan-chidambaram@users.noreply.github.com",
          "name": "Natarajan"
        },
        "message": "updated model for skl=1.5.2",
        "distinct": true,
        "url": "https://api.github.com/repos/natarajan-chidambaram/RABBIT/commits
      }
    ]
  },
  "public": true,
  "created_at": "2024-10-16T14:14:10Z"
},
```

Figure 2.6: JSON output of a PushEvent obtained from the REST API `events` endpoint for GitHub contributor natarajan-chidambaram.

In this thesis, we predominantly use the `events` endpoint of the REST API to query the events performed by contributors in GitHub. This endpoint can be accessed using the query: `https://api.github.com/users/<username>/events`, and the following query can be used for accessing the events performed in a repository: `https://api.github.com/repos/<repo_ownner>/<repo_name>/events`. Every event has these common fields: unique event "id", the "type" field that provides the event type that is being performed, the "actor" field that provides details regarding the contributor that performed the event, the "repo" field that provides information regarding the repository in which the event is performed, and the "payload" field that provides additional details regarding the event. In total, the `events` endpoint reports

17 event types[16] such as IssuesEvent when a contributor opens/closes/reopens an issue in a repository, PullRequestEvent when a contributor opens/closes/reopens a PR in a repository, IssueCommentEvent when a contributor posts a comment under an issue or a PR, ReleaseEvent when a contributor creates a release in a repository, and so on.

```
{
  "id": "42908234855",
  "type": "CreateEvent",
  "actor": {
    "id": 48755692,
    "login": "natarajan-chidambaram",
    "display_login": "natarajan-chidambaram",
    "gravatar_id": "",
    "url": "https://api.github.com/users/natarajan-chidambaram",
    "avatar_url": "https://avatars.githubusercontent.com/u/48755692?"
  },
  "repo": {
    "id": 721712501,
    "name": "natarajan-chidambaram/RABBIT",
    "url": "https://api.github.com/repos/natarajan-chidambaram/RABBIT"
  },
  "payload": {
    "ref": "2.3.1",
    "ref_type": "tag",
    "master_branch": "main",
    "description": null,
    "pusher_type": "user"
  },
  "public": true,
  "created_at": "2024-10-16T14:24:13Z"
},
```

Figure 2.7: JSON output of a CreateEvent obtained from the REST API `events` endpoint for natarajan-chidambaram.

Fig. 2.6 provides the output returned by the `events` endpoint for a PushEvent and Fig. 2.7 represents a CreateEvent. From the “actor” field, we can observe that both events are performed by the same contributor with unique “id” 48755692 and unique “login” name natarajan-chidambaram. From the “repo” field, we can observe that both events are performed in GitHub repository with unique “id” 721712501 and unique repository “name” *natarajan-chidambaram/rabbit*. However, the “payload” field has different information as it depend on the event type. Fig. 2.6 provides the size of the commit being pushed, the hash values before and after pushing the commit, and commit details such as its author and commit message. On the other hand, the “payload” field in Fig. 2.7 gives details regarding the tag that is created in the repository, its “ref” name, description and so on.

2.8. Summary

GitHub provides various automation mechanisms that developers and practitioners can use to automate their repetitive, error-prone and effort-intensive tasks. Automa-

[16]`https://docs.github.com/en/rest/using-the-rest-api/github-event-types?apiVersion=2022-11-28`

tion mechanisms can be triggered by a wide variety of events. The choice of automation mechanism depends on the task that needs to be automated. GitHub workflows are predominantly used for performing CI/CD tasks. Internal automation services are often used for checking dependency and security vulnerabilities in projects, Apps can be installed and bot accounts can be granted permission in repositories for performing various activities. Finally, external automation services perform tasks in an external system based on the events happening in repositories. Automation mechanisms can interact with repositories using the GitHub API. Activities performed by automation mechanisms will be visible through the GitHub UI under the name of their bot actors or bot accounts which are collectively called as bots.

This thesis specifically focuses on identifying the difference in activities between bots and humans. It could also be of interest to study the behaviour and usage of automation mechanisms such as GitHub workflows and Actions, but it is out the scope for the current thesis as it is being studied by other researchers at our lab (Decan et al., 2022; Rostami Mazrae et al., 2023; Decan et al., 2023; Onsori Delicheh et al., 2024).

CHAPTER 3

State of the art

Software developers and maintainers in GitHub repositories use bots to assist them in performing error-prone and repetitive activities. As seen in Section 1.4, bots are prevalent in GitHub repositories. Their usage and behaviour may differ depending on the activities they automate, the environment they operate in and so on. Many studies regarding bots have been conducted in the past. This chapter provides an overview of the research on using bots in collaborative software development. Section 3.1 reports on existing empirical studies to categorise bots in GitHub, identifying challenges in using them, and analysing their impact in software development. Section 3.2 provides an overview of bot identification tools and approaches that researchers have created and used for identifying bots in GitHub.

3.1. Studies on bots

This section provides an overview of existing studies related to bots. Section 3.1.1 outlines different categories of bots, Section 3.1.2 highlights the challenges faced with bots and their impact in software projects, and Section 3.1.3 provides an overview of various empirical studies involving bots that were conducted in the past.

3.1.1 Categorising bots

Erlenhov et al. (2020) categorised bots based on three personas namely autonomous bots, chat interfaces and smart bots. They carried out a qualitative study focussing on the definition of what is a bot, why developers use them and what developers struggle with when using bots. The findings of this study were reported based on interviews with 21 developers and surveys with 111 developers.

Bots with an *autonomous persona* work on their own without requiring much human intervention to complete certain tasks that humans would do. These bots could improve productivity of developers. Bots with a *chat persona* (commonly referred to as chatbots) communicate with developers through a natural language interface (voice or chat). These bots usually provide an interface to an existing system, and do not constitute a full-fledged system by themselves. Such bots use existing communication tools (e.g., Slack and Discord) that human developers use for status updates and synchronisation on work tasks. Bots with a *smart persona* are adaptive at executing certain tasks. These bots are often strongly associated with machine learning and advanced program analysis techniques (e.g., DeepCode AI[1] and GitHub Copilot.[2])

Wyrich & Bogner (2019) proposed a smart and autonomous bot that integrates into the team like a human developer via the existing version control platform. The bot automatically performs refactorings and presents the changes to a developer for asynchronous review via PRs enabling the developers to review it anytime.

Lambiase et al. (2024) conducted a multivocal literature review using 79 formal literature (e.g., published in journals and conferences) and 28 grey literature (e.g., blog posts and white papers) to provide a taxonomy for categorising bots. For formal literature, they relied on Scopus,[3] IEEE Xplore[4] and ACM digital library,[5] and for grey literature, they used the Google search engine. They identified four main categories. *Software product bots* perform automatic operations on source code or related artifacts and fall under the following five sub-categories: development, refactoring, testing, configuration and CI/CD. *Software process bots* are used to improve and enhance communication, collaboration and management activities and belong to four sub-categories: team, communication and collaboration, task, and code review. *Knowledge bots* store, share and manage information and knowledge about software projects and belong to documentation and metrics sub categories. *Emergent bots* are

[1]`https://snyk.io/platform/deepcode-ai/`
[2]`https://github.com/features/copilot`
[3]`https://www.scopus.com/home.uri`
[4]`https://ieeexplore.ieee.org/Xplore/home.jsp`
[5]`https://dl.acm.org/`

emerging bots that are still in an embryonic stage and belong to three sub-categories: digital twin bots, orchestration bots and technology transfer bots.

Ghorbani et al. (2023) qualitatively analysed the characteristics affecting developer preferences for interacting with bots in PRs. They formulated 13 questions to collect data based on contributor experiences of using bots in their respective communities. By interviewing 12 participants from four different open-source communities they identified seven themes on how software developers perceive software bots: attitude, autonomy, persona, task, feelings, project norm, and role. Among these, they found autonomy and persona to exert more influence in shaping developer perception of bots. To further study this influence of autonomy and persona, they conducted surveys among 56 participants and recommended that developers should have options to scale the autonomy of bots, select and change bot personas, and improve project-specific feedback on bot behaviour and developer preferences.

Wessel et al. (2018) analysed the usage of 48 different bots in 93 GitHub projects and categorised bots into 12 different categories based on the tasks they perform in software projects. Those tasks include reviewing PRs, running automated tests, building projects, analysing and updating dependencies, and creating issues. To analyse the changes in PR characteristics before and after bot adoption, they looked into PRs belonging to 44 projects for a duration of six months before and six months after bot adoption. They found statistically significant differences in 44 GitHub projects in terms of number of commits, number of changed files, number of comments and so on before and after bot adoption.

Following this, Wang et al. (2022) identified 201 bots in 613 GitHub projects and grouped them into six categories based on their tasks: CI assistance, issue and PR management, code review assistance, dependency and security analysis, developer and user community support, and documentation generation. They identified that 60% of the projects that they considered use at least one bot to automate routine tasks and 74 out of 201 bots belong to more than one category.

3.1.2 Challenges and impact of bots

Based on 21 interviews with software developers, (Erlenhov et al., 2020) reported the challenges faced by developers in using bots. They categorised the identified challenges into three groups: interruption and noise, trust, and usability. Bots that create *Interruption and noise* are the ones that overload human communication channels (e.g., Slack) with too much information or posting so frequently that human developers stop paying attention. Bots that cause *trust* issues to developers are the ones that produce too many broken builds, failed deployments, or provide irrelevant warnings. *Usability* of a bot becomes a challenge when developer has to remember what exactly to type to trigger which functionality.

Saadat et al. (2021) examined the event sequences of repositories with and without bots using a contrast motif discovery method to detect subsequences that are more prevalent in one set of event sequences versus the other. They concluded that teams using bots are more likely to intersperse comments throughout their coding activities, while not actually being more prolific commenters. Also, they suggested that it is crucial to further study the performance of teams that combine human and bot contributors to ensure that continuous communication compensates for the loss of situation awareness and does not negatively impact the performance of teams.

To better understand the impact of bots in GitHub, Wessel et al. (2021) interviewed 21 practitioners and identified 25 challenges that development bots bring to software projects, categorised into three categories: *interaction challenges* (e.g., intimidating newcomers, providing non-comprehensive feedback and impersonating developers), *adoption challenges* (e.g., burden to set up configuration files, limited configurations and handling technical failures) and *development challenges* (e.g., building multitasking bots, restricted bot actions by GitHub API, and hosting and deploying bots). Also, the authors mention some annoying bot behaviours which can be perceived as noise including verbosity, too many actions, and unrequested or undesirable tasks on PRs.

Lambiase et al. (2024) extended this study by conducting a multivocal literature review (as mentioned in Section 3.1.1) and identified more sub-categories on challenges of bots for software engineering purposes. To *interaction challenges* they added long response latency, lack of guidelines for usability of bots, unsatisfied expectations, natural language understanding and processing (e.g., incorrect interpretation, unexpected bias such as text limited to a specific culture and background of bot developers), and uncanny valley which is the phenomenon where a software program behaving like a human being evokes a sense of unease for the person interacting with it. In *adoption challenges* they included lack of trust in recommendations and AI-powered bots, and terms of service. To *development challenges* they added difficulty in identifying the correct bot development framework (e.g., Google Dialogflow, Amazon Lex, Microsoft Bot Framework), lack of tests and difficulty in monitoring the impact of maintenance activities, difficulty in designing architecture for such complex systems, difficulty in integrating different technologies, and difficulty in effectively training AI models. Further, they suggested possible best practices for overcoming these challenges and divided them into two categories based on their scope. They are *development and design* (e.g., follow a modular architecture, make bots adaptive and able to learn over time using AI, and adopt a specific lifecycle process for bots), and *interaction and adoption* (e.g., allow developers to edit bot configuration easily, provide bots with personality, and enforce transparency in bot actions and outputs).

Wessel et al. (2023b) surveyed 205 open source contributors and 23 maintainers on challenges of using and interacting with bots, and interviewed 21 practitioners who are experienced with bots, including project maintainers, contributors, and bot developers. According to the authors, intelligence and adaptability are not yet widely present in bots that work on GitHub, although they recurrently appear in the literature as desired bot characteristics. So, based on their analysis on the developers' and maintainers' needs and expectations from a bot, they proposed seven guidelines for both bot developers and tool builders, which are: (1) provide clear, concise, and well-organised information, (2) focus on an appropriate way to show information, (3) provide actionable changes to developers, (4) avoid overly humanised bot messages, (5) make bots' purpose clear, (6) provide options to configure bot notification, and (7) include documentation of alternative installation settings to accommodate different types of users. As noise emerged as a central interaction challenge, they investigated how to overcome it and proposed two strategies: (1) create a mediator bot that organises existing bot information in a PR, and (2) create a separate interface for bot interaction in PRs. In a follow-up work, Wessel et al. (2022a) interviewed 32 practitioners experienced with OSS bots, presented a fictional story of a mediator bot capable of better supporting developers and found 22 design strategies for creating such a bot. The aim of such a mediator bot was to summarise and customise other bots' actions to reduce information overload incurred by their use. They grouped the identified design strategies into four categories for developing such a mediator bot and one category for modifying GitHub's interface: (a) information management (e.g., summarisation of bot comments); (b) newcomer assistance (e.g., sending them a welcoming message); (c) notification management (e.g., schedule bot notifications); (d) spam and failure management (e.g., bug reports); and (e) platform support (e.g., separating bot comments). Ribeiro et al. (2022) followed the above-mentioned design strategies and developed a prototype of mediator bot called FunnelBot. They qualitatively evaluated the performance of FunnelBot among 25 participants and found that it is appropriate, clear, easy to understand, useful, and organises information very well compared to that of using multiple bots.

3.1.3 Empirical studies

In the past, there have been many empirical studies on bots (Rebai et al., 2019; Wessel et al., 2019; Romero et al., 2020; Dey et al., 2020c; Wyrich et al., 2021; Zhang et al., 2022; Wessel et al., 2022b; Kazi Amit et al., 2023; Khatoonabadi et al., 2023; Mohayeji et al., 2023; Fischer et al., 2023; Murali et al., 2024). In this section, we focus on recent studies that have been conducted.

Wyrich et al. (2021) analysed the PRs created and commented by humans and bots to understand the difference in priority given to PRs created by bots and humans. They found that PRs created by humans received faster response and 73% of them were merged. PRs created by bots took significantly more time to receive a response and only 37% of them were merged, even though they contained fewer changes on average than PRs of humans.

On the other hand, Kazi Amit et al. (2023) studied the time-to-first-response for PRs by analysing 111,094 closed PRs from ten popular OSS projects on GitHub. To understand the properties of a PR, the authors identified 24 features belonging to three dimensions namely *PR* (e.g., length of PR description, tag exists?, and #commits in PR at opening time), *project* (e.g., # active core team members in the last three months, #PRs, and #executable lines of code), and *developer* (e.g., #previous PRs, #PR reviews in a project, and core member?). They found around 80% of PRs to get their universal first response within a day, 40% of PRs that received their first response on same day of PR creation date within 10 minutes, of which 70% were by bots (CLA bots, review supporting bots and reviewable services). In summary, they found that bots frequently generate the first response in PRs, and there is a significant time difference between the first response provided by a bot and first response provided by a human in a PR.

Zhang et al. (2022) quantitatively studied the factors influencing PR latency and the change in these factors with a change in context (e.g., time, project and developer). They identified 47 features that influence PR latency. They classified these features into developer characteristics (e.g., *Is this the developer's first PR?* and *Do contributor and integrator have the same affiliation?*), project characteristics (e.g., team size and project age), and PR characteristics (e.g., whether it is bug fix, and the number of PR comments). As they found a widespread usage of bots in PRs, they conducted a case study to identify the impact of bots on PR latency. Executing an existing bot identification tool to identify bots in 3.3M+ PRs belonging to 11K+ GitHub projects obtained from GHTorrent, they observed that more than one out of three PR comments were made by bots. Further, they found that the presence of comments posted by humans were more important than those of bots in explaining PR latency.

Wessel et al. (2022b) qualitatively analysed the unexpected impacts of adopting a code review bot in 1,194 software projects and interviewed 12 practitioners including open source maintainers and contributors. Through analysis, the authors reported that the number of monthly merged PRs increased after the introduction of a code review bot, the number of monthly non-merged PRs decreased after the introduction of a code review bot, and communication among developers decreased.

On the other hand, Khatoonabadi et al. (2023) qualitatively studied the potential benefits and drawbacks of using stale bot for pull-based development. Stale bot is a GitHub Action[6] that automatically warns and closes issues and PRs that have been inactive for a certain period of time. By observing PRs for a duration of two years (1 year before and 1 year after adopting stale bot) in 20 large and popular open-source projects, they concluded that stale bot can help projects deal with a backlog of unresolved PRs and also improve the PR review process, but may negatively affect project newcomers.

Mohayeji et al. (2023) carried out a fine-grained analysis on the lifecycle of vulnerabilities to manifest how they are dealt with in the presence of Dependabot (GitHub's internal automation service for checking dependencies, presented in Section 2.3). On one hand, they found many projects to use Dependabot to fix vulnerable dependencies and security updates that are merged after several days. On the other hand, when developers do not merge a security update, they usually address the identified vulnerability manually. This approach, however, often takes up to several months which in turn could expose the projects to security issues.

To further study the time for merging these security updates Fischer et al. (2023) performed time-series analysis on security-altering commits in GitHub. They show that while all of GitHub's security interventions have a significant positive effect on security, they differ greatly in their effect size. Also, they studied the design of each intervention to identify the building blocks that worked well and those that did not.

Murali et al. (2024) studied diversity in issue assignment between bot and human assignors. By analysing 127K issues that have assignees in VSCode, Tensorflow and Kubernetes projects, they found that bots are more biased than humans in a majority of cases, and in some cases humans are extremely biased in issue assignments. Also,

[6] `https://github.com/actions/stale`

they conclude that it is possible to design unbiased issue assignment bots as they found bots in Kubernetes to be highly competitive with humans in terms of diversity and outperform humans on many occasions.

3.2. Bot identification in GitHub

The previous section reported on studies that were conducted based on the usage, behaviour and impact of bots in GitHub repositories. To perform such studies, bots need to be identified from a set of contributors chosen for observation. Section 3.2.1 provides various bot identification heuristics, tools, and models that researchers have proposed. Section 3.2.2 provides an overview of studies that relied on bot identification approaches.

3.2.1 Approaches to identify bots

The most naive approach is a Name-Based Heuristic (NBH) (e.g., sre-bot, arduinobot, ezrobot). It is based on simple regular expressions to detect whether a contributor name contains specific substrings (e.g., "bot", "automate", or "robot"). This heuristic has been used with different variations to identify bots in various studies (Murali et al., 2024; Orrei et al., 2023; Schueller et al., 2022; Dey et al., 2020a)

Dey et al. (2020a) developed BIMAN, an ensemble model that combines three different approaches to recognise bots in commits. The first approach is a variation of NBH, called *BIN* (for Bot Identification by Name) that uses a regular expression that checks for the "bot" substring in contributor name preceded and/or followed by non-alphabetic characters (e.g., sre-bot, github-actions[bot]). The second model called *BIM* (for Bot Identification by commit Message) relies on similarity in commit messages, based on the assumption that the textual variation in commit messages is lower for bots than for humans. The third model called *BICA* (for Bot Identification by Commit Association) is a Random Forest binary classifier trained on six features related to the modifications made to files in commits: (i) number of files changed by commit author across commits; (ii) number of unique file extensions in all commits; (iii) standard deviation of number of files per commit; (iv) mean number of files per commit; (v) number of unique projects commits have been associated with; and (vi) median number of projects commits have been associated with. To train *BIM* and *BICA*, they used a dataset consisting of 26,300 commit authors, of which 13,150 were bot authors and 13,150 were human authors. To create such a dataset, they used *BIN*, then manually verified the author IDs, description, commits messages and PR comments whenever available to correct their type. Evaluating BIMAN after training resulted in an AUC-ROC score of 0.90.

Golzadeh et al. (2021b) developed BoDeGHa, a model and associated open-source tool that uses a Random Forest binary classifier to identify bots that are involved in commenting issues and PRs. BoDeGHa takes as input a repository name and a GitHub API key to provide a prediction for each contributor that posted comments under issue or PR in that repository. For each contributor, BoDeGHa retrieves their last 100 issue and PR comments and computes five features: (i) the string distance between comments (using a combination of Levenshtein and Jaccard distance), (ii) the number of comment patterns (sets of very similar comments), (iii) the Gini inequality between comment patterns, (iv) the total number of comments, and (v) the number of empty comments. They created a manually labelled ground-truth dataset of 5,000 GitHub contributors of which 527 were bots. They used 60% of contributors for training (316 bots and 2,684 humans) and the remaining 40% (211 bots and 1,789 humans) for testing the model. Evaluating the trained model on test data resulted in precision, recall and F1-score of 0.98 each.

In a follow-up work, they developed BoDeGiC (Golzadeh et al., 2020) that uses a Random Forest binary classification model to identify bots that are involved in git commits. BoDeGiC takes the same arguments as inputs but uses an approach that is similar to the one of BoDeGHa. The former is applied to commit messages rather than issues and PR comments. For each contributor, BoDeGiC retrieves their last 100 commit messages and computes the latter four features that were used in BoDeGHa (i.e., it does not include string distance between commit messages as a feature). To train the classification model, they relied on the dataset of git commits that was used by Dey et al. (2020a). They obtained a dataset that consists of 311,622 git commit messages from 6,922 contributors (3,380 bots and 3,542 humans). Evaluating BoDeGiC on the test set consisting 40% of contributors (1,352 bots and 1,417 humans) resulted in precision, recall and F1-score of 0.80 each.

Abdellatif et al. (2022) developed BotHunter, a state-of-the-art bot identification approach. It relies on a Python script that executes a Random Forest binary classification model to identify bots in GitHub. BotHunter takes as input either the login name of a contributor or the name of a repository for which it obtains the login name of contributors (through GitHub REST API's `contributors` endpoint) who made at least one commit to the repository. Then for each contributor, it queries the GitHub API to retrieve their data. To identify bots, BotHunter uses 19 features, three of which are in common with BoDeGHa and BIMAN (similarity in text between issue/PR comment, commit message and comments that precede bot events in issues and PRs), six are based on profile information (account login, name, tag and bio, number of followers and number of followings) and 10 are based on account activity (total number of repositories, issues, PRs and commit events, unique number of repositories, issues, PRs and commit events, median events per day and median response time to the earliest event in issue or PR). Among the considered features, the top five most important features were *account name*, *account login*, *number of followers*, *issue/PR comments similarity*, and *median events per day*. To train and evaluate the classification model, they combined the dataset used for BIMAN and BoDeGHa and got a set of 669 bots and 4,428 human contributors in GitHub. Evaluating the trained classification model resulted in a precision of 0.957, recall of 0.894 and F1-score of 0.924.

In another work, Golzadeh et al. (2021a) proposed a multinomial Naive Bayes classification model that can provide the probability that a comment is created by a bot. It works at the level of individual comment in issues and PRs. It uses TF-IDF, an NLP technique, where a feature vector is formed for each comment. To train the classification model, they relied on the dataset published in earlier work (Golzadeh et al., 2021b) from which they took a subset of 9,641 comments created by 519 bots and 9,641 comments created by 4,090 humans. Evaluating this model on the test set of 9,515 comments, they achieved a precision, recall and F1-score of 0.882 each.

Cassee et al. (2021) developed three comment-level classification models for identifying bots through their issues and PR comments. The first model was a Naive Bayes classifier that uses TF-IDF to classify comments based on its text. The second model was a Support Vector Machines binary classifier that uses average Jaccard distance and normalised Levenshtein distance between comments to capture templated comments. The third model was a Random Forest binary classifier that uses four features regarding the activity of the contributor that authored the comment: (i) number of repositories created, (ii) number of gists created, (iii) number of users followed by the contributor, and (iv) number of users that follow the contributor. To train these models they relied on the contributor names provided by Golzadeh et al. (2021b), under-sampled the number of human contributors to balance the dataset, and used the GitHub REST API to gather 8,528 comments from 393 bots and 8,607 comments from 406 human contributors. As in other studies, they evaluated their model on test data and computed the precision, recall and F1-score. The first model achieved a performance of 0.882 each, the second model achieved 0.918 each and the third model achieved 0.962 for each metric.

3.2.2 Studies that relied on bot identification approaches

Bot identification has been used by various researchers for different purposes such as to treat bot activities differently from those of humans, to remove bot activities from the study and so on. This section reports on how various bot identification approaches have been used in the research literature.

As explained in Section 2.3 and Section 2.4, GitHub tags the bot actors (on behalf of GitHub Apps and internal automation services) as 'bot' in the GitHub UI. As presented in Section 1.3, the "type" field in `users` endpoint for a bot actor will have "Bot" as its value and GitHub adds a specific string '[bot]' at the end of the bot actor's name (e.g., renovate[bot]) GitHub prevents regular user login names to use the characters '[' and ']'. Some studies (Wyrich et al., 2021; Moharil et al., 2022; Orrei et al., 2023; Khatoonabadi et al., 2024) included this approach to identify bot contributors.

While NBH has been used in some studies (Murali et al., 2024; Orrei et al., 2023; Schueller et al., 2022; Saadat et al., 2021; Khatoonabadi et al., 2024), with the latter two performing a manual inspection on the results, more sophisticated variations of this approach have been used to identify bot contributors. Wyrich et al. (2021) used '%bot', '%robot', '%-bot-%', '%-robot-%', 'bot-%' and 'robot-%' to identify bots. Lin et al. (2024) considered 'bot' suffix in contributor names, whereas Fang et al. (2023) used '-bot' or '-robot' as suffix of the contributor names to identify bots. To develop the *BIN* model of BIMAN, Dey et al. (2020a) relied on NBH to check for 'bot' preceded or followed by non-alphabetic characters. Abdellatif et al. (2022) included NBH as a feature in their model to identify bot contributors by relying on presence of substring 'bot' or 'automate' in the login name, bio or description of the contributor. For the contributors that could not be associated to a GitHub account, Schueller et al. (2022) extracted their email ID and filtered it based on a suffix that is equal to 'bot', 'ghbot', 'bors' and 'travis'.

Some studies make use of existing ground-truth datasets that were published to identify bots in GitHub. Many studies (Lin et al., 2024; Schueller et al., 2022; Orrei et al., 2023) relied on the pre-defined list of bots published by Golzadeh et al. (2021b), whereas as Fang et al. (2023) relied on the bot dataset used by Dey et al. (2020a), and Khatoonabadi et al. (2024) used bots listed by Golzadeh et al. (2021b), Wang et al. (2022) and Abdellatif et al. (2022).

Also, the bot identification approaches presented in Section 3.2.1 have been used by some studies to identify bot contributors. BoDeGHa was used by Bock et al. (2023) on contributors that made at least 2,000 issue/PR comments, and Moharil et al. (2022) executed BoDeGHa on contributors that made at least one issue comment. Further, Zhang et al. (2023) used BoDeGHa with a small modification to predict the type of contributor by accepting PR comments provided by the user rather than extracting the data from the GitHub API.

Finally, Schueller et al. (2022) and Fang et al. (2023) ordered the contributors based on their number of commits and manually checked the top 100 contributors to identify more bots in their collection of considered bots.

CHAPTER 4

Improving existing bot identification approaches

This chapter addresses **Goal 1** of this thesis. We leverage the predictions provided by existing bot identification approaches to propose improved models that detect bots with better performance. Section 4.1 is based on our publication (Golzadeh et al., 2022a) that presents an exploratory study on the performance of existing bot identification approaches and highlight that bots are among the top contributors in certain repositories. Then, we propose and evaluate a new ensemble model that combines the prediction provided by multiple bot identification approaches to provide a more accurate final prediction. Section 4.2 is based on our publication (Chidambaram et al., 2022) that presents empirical results on the predictions made by BoDeGHa on a set of contributors and highlight its limitations. To overcome these limitations, we introduce a model that improves the contributor type predictions by leveraging the predictions provided by BoDeGHa across multiple GitHub repositories.

4.1. An ensemble bot identification model

In Section 1.4, bots have been shown to belong to the top contributors in certain repositories. Their prevalence is challenging for researchers conducting quantitative socio-technical analyses on software repositories since neglecting the presence of bots might lead to incorrect and misleading conclusions during empirical analyses as mentioned in Section 1.5. The ability to identify bots is also important for empirical studies about the role played by bots in collaborative software development (Wessel et al., 2018). Last but not least, communities and funding organisations can benefit from bot identification tools to correctly recognise, accredit and sponsor human activity (Hauff & Gousios, 2015). This section addresses a part of **Goal 1** of this thesis. Section 4.1.1 provides the motivation and research questions. Section 4.1.2 quantifies the number of repositories, bots and humans that are used in this study. Section 4.1.3 details on the training and evaluation of a new bot identification model that combines the predictions of multiple existing bot identification approaches. Section 4.1.4 makes use of this new ensemble model to identify the prevalence of bots among the active contributors in the selected repositories. Finally, Section 4.1.5 compares the proportion of commits made by bots and humans in each considered repository.

4.1.1 Motivation

Section 3.1 reported that software developers and repository maintainers use various automation mechanisms to reduce their workload and increase productivity in GitHub. The prevalent presence and activity of bots in software repositories (as highlighted in Section 1.4) makes it challenging for software engineering researchers to study socio-technical aspects of software development since their findings may be biased by not explicitly considering the presence of bots among the contributors (Golzadeh et al., 2021b). Similarly, it may be important for contributors that their contributions are properly recognised and rewarded since collaborative software development activities are often considered as a criterion for employers when hiring developers (Hauff & Gousios, 2015). This is especially important when funding or donations are awarded to contributors based on their contributions. While there are tools such as *SourceCred*[1] to support communities in automatically measuring and rewarding value creation, they do not automatically identify bots and their activities so far. This is where the bot identification approaches presented in Section 3.2.1 come to the rescue.

This section is based on my co-authored publication in the International Workshop on Bots in Software Engineering (Golzadeh et al., 2022a), and presents an exploratory study on the accuracy of five bot identification approaches on a set of 540 contributors from 27 GitHub repositories. We show how prevalent bots and their activities are, and that none of the existing bot identification approaches are accurate enough to detect bots even among the most active contributors. We also show that combining these approaches increases the precision and recall of bot identification but remains

[1] `https://sourcecred.io`

insufficient to capture all bots and their activities. This lack of identifying all bots highlights the need for improved bot identification approaches.

4.1.2 Ground-truth dataset of contributors

In this section, our goal is to empirically evaluate the accuracy of bot identification approaches. To do so, we need to have a dataset of active software development repositories with a large number of commits and contributors along with their ground truth (i.e., *bot* or *human*.) We relied on the SEART GitHub search tool (Dabic et al., 2021) to query candidate repositories that have at least 100 contributors, were not fork and had been active in October and December 2021. From these, we randomly selected 27 large and active repositories that have at least 1,200 commits and 200 contributors. In total, the 27 selected repositories account for 175,499 commits from 9,426 contributors and cover a wide variety of programming languages (e.g., Javascript, Python, Java, PHP, Ruby, Rust, Go) and software domains such as software development packages, plugins, and tools.

For each repository, we queried the GitHub REST API's `contributors` endpoint to retrieve the 20 most active GitHub contributors in terms of commits, and their respective number of commits. The resulting dataset consists of 540 contributors. To obtain the ground-truth of contributors, two researchers (co-authors of the publication Golzadeh et al. (2022a)) manually and independently followed the guidelines given for this process and determined the type of all the considered contributors. For each contributor, the researchers checked their (1) GitHub profile, (2) commit and commenting activity, and (3) activities through `events` endpoint. During this process, we found 50 bots out of the 540 considered contributors, which will be used as a ground-truth for deciding on the contributor type.

4.1.3 Ensemble classification model

Bot identification approaches. In this section, we evaluate the accuracy of the following five bot identification approaches:

1. GitHub account type. This approach relies on the GitHub REST API's `users` endpoint to determine whether a given GitHub contributor is a bot actor. If the value of the "type" field in the JSON response provided by the `users` endpoint is "Bot", then we can conclude that the contributor corresponds to a bot actor (belonging to either a GitHub App or internal automation service) as explained in Section 1.3.

2. "bot" suffix. This approach relies on the presence of the string "bot" at the end of the author's name. It has been used by several research studies, as explained in Section 3.2.2.

3. List of bots.[2] This approach checks the contributor name against a predefined list of names of 527 known GitHub bot contributors that were manually identified

[2]`https://doi.org/10.5281/zenodo.4000388`

by Golzadeh et al. (2021b) among 5,000 GitHub contributors.

4. BoDeGHa.[3] Golzadeh et al. (2021b) proposed a classification model and tool to identify bots in GitHub PR and issue activity. Their model is based on features extracted from issue and PR comments.

5. BoDeGiC.[4] Golzadeh et al. (2020) further extended the above approach to support git commit messages, and implemented the resulting model as part of a tool named BoDeGiC.

We applied the five bot identification approaches on our dataset of 540 contributors. Fig. 4.1 shows the classifications provided by these approaches. For readability, we only report on the 87 contributors that either correspond to a bot according to the ground-truth dataset created in the previous section, or that were classified as *bot* by at least one of the approaches. Contributors labeled as bot in the ground-truth are shown on the left side of the vertical blue line. Contributors labeled as human in the ground-truth are shown on its right. An orange cell indicates that the contributor was identified as a bot by the corresponding approach, while a blue cell indicates that it was identified as a human contributor. Grey cells correspond to cases where the approach is unable to determine the contributor type. In the case of BoDeGHa, this corresponds to contributors with less than 10 comments in PRs or issues. In the case of BoDeGiC, this corresponds to contributors having less than 10 commits made with a committer name matching their GitHub login name.

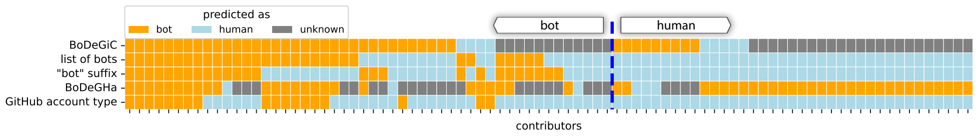


Figure 4.1: Classifications (*bot*, *human* or *unknown*) obtained from the five bot identification approaches. Only bots according to the ground-truth (at the left of the vertical blue dashed line) and humans wrongly classified as bot (at the right of the vertical blue dashed line) are displayed.

From this figure, we observe that list of bots, "bot" suffix and GitHub account type are safer approaches, in the sense they do not wrongly classify human contributors as bots. At the same time, they missed many actual bots: from 19 for list of bots to 32 for GitHub account type. We also observe that BoDeGiC effectively captures 34 bots, but at the same time, wrongly considers nine human contributors as bots. BoDeGHa exhibits a similar behaviour: it is able to capture 25 out of 50 bots, but wrongly misclassifies 30 humans as bots. We note that none of the approaches is perfectly effective in detecting bots except for four cases, where the five approaches do not even agree on whether a given contributor is a bot or not. However, only four

[3]https://github.com/mehdigolzadeh/BoDeGHa
[4]https://github.com/mehdigolzadeh/BoDeGiC

Table 4.1: Recall (R), precision (P) and $F1$-score (F1) of bot identification approaches (in ascending order of bot recall).

	bots			**humans**			**overall (weighted)**		
approach	R	P	$F1$	R	P	$F1$	R	P	$F1$
Baseline ZeroR	0.0	0.0	0.0	1.0	.907	.951	.907	.823	.863
GitHub account type	.360	**1.0**	.529	**1.0**	.939	.968	.941	.944	.928
BoDeGHa	.500	.455	.476	.939	.948	.944	.898	.903	.900
“bot” suffix	.520	**1.0**	.684	**1.0**	.953	.976	.956	.958	.949
List of bots	.620	**1.0**	.765	**1.0**	.963	.981	.965	.966	.961
BoDeGiC	.680	.791	.731	.982	.968	.975	.954	.951	.952
EnsBoD	**.900**	.865	**.882**	.986	**.990**	**.988**	**.978**	**.978**	**.978**

contributors labeled as bot in the ground truth are not detected as such by any of the approaches (shown before the vertical blue dotted line in Fig. 4.1), suggesting that a combination of the approaches could lead to an improved bot identification model.

We build such an ensemble model called EnsBoD, by training a decision tree classifier taking as input the classifications made by each of the five approaches and providing a final decision whether the corresponding contributor is *bot* or *human.* We trained and validated EnsBoD following a 10-fold cross-validation process.

Table 4.1 reports on the precision, recall and F1-score of each approach applied on the whole dataset of 540 contributors, distinguishing these scores between bot and human contributors. Since the dataset has a fairly imbalanced number of human and bot contributors and for completeness, we also report on the overall weighted scores by attributing a class weight inversely proportional to the number of cases. Given there are far more human contributors than bot contributors in the dataset, these high scores (between 0.898 and 0.966) are mostly driven by the scores obtained for human contributors. To ease the interpretation of these scores, we also provide the scores for a baseline ZeroR model classifying all contributors as human contributors (i.e., the majority class).

The observations that can be made from Table 4.1 for the existing bot identification approaches match the ones we made from Fig. 4.1. In particular, we observe that some approaches (namely GitHub account type, “bot” suffix and list of bots) have a perfect precision but are not able to capture as many bots as BoDeGiC. This should not come as a surprise. For example, GitHub account type has no false positive since it is impossible for a human contributor to flag his/her own user account as a bot corresponding to a GitHub App or internal automation service. Similarly, list of bots relies on a predefined list of bot names that were manually validated by a group of researchers. On the other hand, the precision reached by “bot” suffix is surprisingly higher than what was reported by Golzadeh et al. (2021b). Depending on the ground-truth contributor type, we found that only around 4% of the contributors having “bot” in their name actually correspond to human contributors.

The mean scores we obtained for EnsBoD are reported in the last row of Table 4.1. Even if it was trained and validated on a small dataset, the EnsBoD model already outperforms any of the five other approaches, with an average recall of 0.900 and an average precision of 0.865 for bots. In Section 4.1.4, we empirically assess the prevalence of bots among contributors, by relying on EnsBoD.

4.1.4 Prevalence of bots among active contributors

Section 4.1.1 underlined the importance of detecting bots in software repositories, not only for researchers aiming at quantifying and understanding their impact on the development process, but also for properly recognising and rewarding contributions made by human contributors. This section aims to quantify the prevalence of bots among the 20 most active contributors in the 27 considered repositories.

We applied EnsBoD on each of the 540 contributors of our dataset to quantify how many of them can be detected. Fig. 4.2 shows the output of EnsBoD for each repository (x-axis) and each contributor (y-axis) sorted by the number of commits they made in the repository. In complement to the output of EnsBoD (i.e., *bot* or *human*), we indicate whether the output is correct ("human user" and "correctly classified bot") or not ("human classified as bot" and "missed bot").

We observe that all the considered repositories are making use of bots, some of them even having four different bots among their 20 most active contributors. Interestingly, many of these bots are responsible for most of the commits in the repositories. For instance, the most active contributor (ranked 1) of six repositories

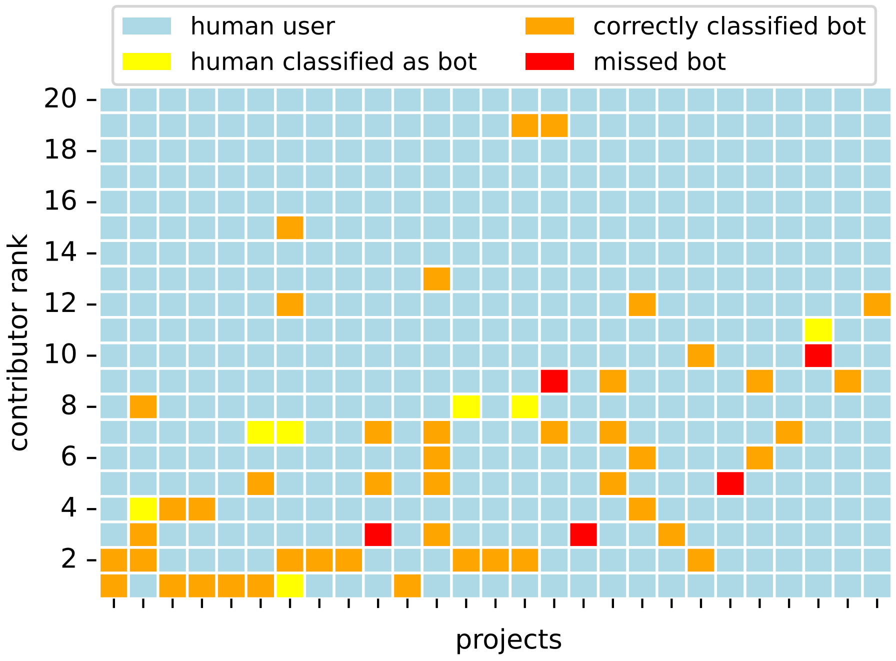


Figure 4.2: Rank in descending order of number of commits made in the repository by top 20 most active contributors in 27 popular open-source software repositories.

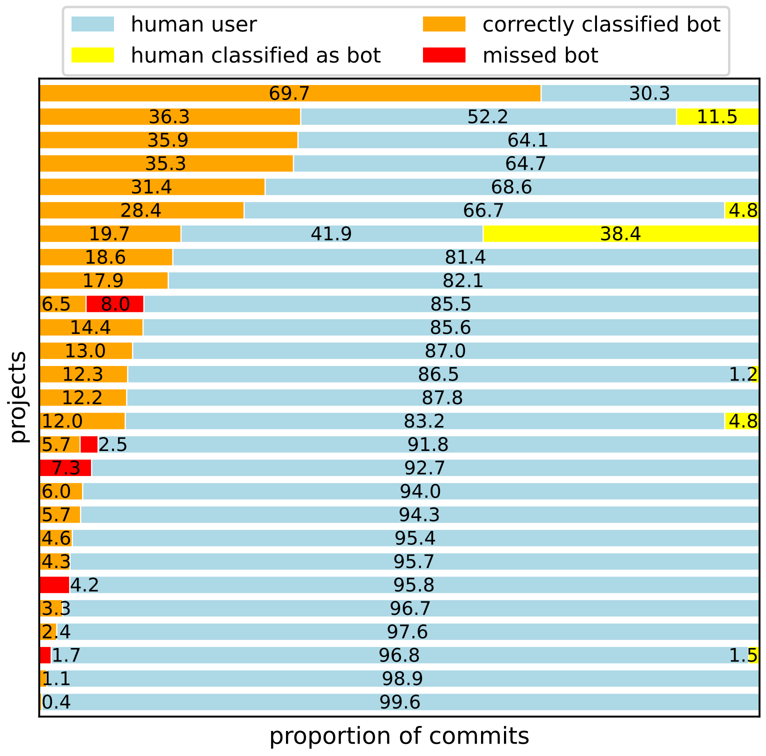


Figure 4.3: Proportion of commits made by the 20 most active contributors in each repository.

is a bot, while 18 out of 27 repositories have a bot in the top three contributors.

We also observe that a non-negligible amount of bots are missed by EnsBoD. For instance, five bots are missed and three of them are among the five most active contributors of the repositories. Similarly, a non-negligible amount of actual human contributors are wrongly classified by EnsBoD: there are seven human contributors that are misclassified as bots, of which one is the most active contributor in the corresponding repository, and five others are within the 10 most active contributors.

These findings show that, while bot identification approaches can help in doing so, even an optimistic combination of them still misses some bots, and still wrongly considers some human contributors as bots.

4.1.5 Presence of bots in commits

This section aims to quantify the proportion of commits made by bot and human contributors in their respective repositories. This is especially important given that tools such as *SourceCred* reward contributors based on their activity, including their commit activity. For each repository, we counted the commits made by each of the

20 most active contributors, distinguishing between bot and human contributors. Fig. 4.3 reports on the proportion of commits made bots and human contributors in each repository.

The figure shows that the commits made by bots represent up to 69.7% of the commit activity. On average, approximately 16% of the commits in these repositories are made by bots (median is 12%), even if bots only account for 9% of the top 20 contributors on average (median is 10%).

While, as observed in Section 4.1.4, EnsBoD is able to detect most of the bots, it still misses some of them, and the missed ones are responsible for 8%, 7.3%, 4.2%, 2.5% and 1.7% of the commits in their respective repositories (i.e., 4.7% on average). On the other hand, EnsBoD wrongly classified seven human contributors as bots, and these contributors were responsible for 38.4%, 11.5%, 4.8%, 1.5% and 1.2% of the commits (i.e., 10.4% on average).

This again underlines the importance of considering bots when analysing commit activity in software repositories, and highlights the need for better bot identification approaches to do so.

4.2. Leveraging predictions in multiple repositories

As seen in Section 3.2, BoDeGHa predicts for each contributor with enough activity in the repository whether this contributor corresponds to a *bot* or a *human*. If a contributor has not made enough comments, BoDeGHa classifies it as *unknown*. This section completes **Goal 1** of this thesis, and is based on my co-authored publication in the International Workshop on Bots in Software Engineering (Chidambaram et al., 2022). This section proposes a new model that improves bot identification by BoDeGHa in GitHub repositories. Section 4.2.1 presents the motivation and research questions addressed in this section. Section 4.2.2 quantifies the number of repositories and contributors considered in this section. Section 4.2.3 quantifies how frequently contributors are active in multiple repositories and Section 4.2.4 quantifies how frequently contributors have different or incomplete predictions. Section 4.2.5 presents the improved GitHub bot identification model to fix different predictions and Section 4.2.6 uses the model to fix incomplete predictions.

4.2.1 Motivation

As BoDeGHa works at the repository level, the same contributor active in multiple GitHub repositories may have different predictions, that is, the contributor may be classified as *bot* in some GitHub repositories and as *human* in some other ones. For example, while BoDeGHa identified the well-known bot actor dependabot[bot] correctly as a *bot* (corresponding to the `Dependabot` internal automation service) in many of the considered repositories (explained in Section 4.2.2), it incorrectly identified dependabot[bot] as a *human* in *artichoke/rand_ mt* repository. This is because the 24 comments made by dependabot[bot] in this repository exhibited 10 different comment patterns, corresponding to a behaviour that is more common for human contributors. At the same time, BoDeGHa classifies the same bot actor as *unknown* in *cossacklabs/themis* repository because it only has 9 comments in this repository.

Similarly, a human contributor can be sometimes classified as a bot. For example, in the GitHub repository *rust-lang/libc* we found a human contributor[5] that is detected as *bot* because most of his/her comments follow a single comment pattern of the form "*bors r+*". This comment is used by a human to instruct the bors bot to accept the PR. On the other hand, this contributor is correctly classified as *human* in *crossbeam-rs/crossbeam* and *rust-lang/rust* repositories for example.

This shows that contributors have different predictions depending on the repository BoDeGHa is applied on. To predict the contributor type irrespective of the repository, we can depend on the wisdom of the crowd principle. More specifically, if one assumes that BoDeGHa is more often correct than wrong in its predictions, then, given a contributor having multiple predictions, we can assume that the most frequent prediction (either *bot* or *human*) is correct, while the least frequent one is not.

The goal of this section is to improve the identification accuracy of BoDeGHa for bots active in issue and PR comments. For this, first we investigate how frequently such situations occur in GitHub repositories. Then we quantify how frequently contributors have different predictions (that is, predicted as *bot* in one repository and *human* in another repository by BoDeGHa), and how frequently they have incomplete predictions (that is, predicted as *unknown* by BoDeGHa). Also, we provide preliminary insights on the wisdom of the crowd principle approach to improve the accuracy of BoDeGHa by leveraging predictions from multiple repositories. Further, we evaluate to which extent different and incomplete predictions can be fixed based on the wisdom of the crowd principle. More specifically, we address the following research questions using the final considered dataset (explained in Section 4.2.2):

- How frequently are contributors active in multiple repositories? We observe that one third of the contributors are active in commenting in multiple repositories.
- How frequently do contributors have different or incomplete predictions? We show that more than half of the contributors identified at least once as bots have different or incomplete predictions.
- To which extent can we fix different predictions made for the same contributor? We show that an improvement of BoDeGHa that integrates the wisdom of the crowd principle is effective at fixing different predictions.
- To which extent can we complete predictions? We show that the same approach is promising to address incomplete predictions.

4.2.2 Dataset

The BoDeGHa bot identification tool takes as input a GitHub repository and outputs whether each contributor in the repository correspond to *bot* or *human*. Since our goal

[5]Name is hidden to comply with GDPR regulations.

Table 4.2: Number and proportion of contributors in function of the number of repositories they are active in.

# repositories →	1	2	3	4 or 5	6 - 9	10+
# contributors	5,671	1,530	496	385	239	211
% contributors	66.5%	17.9%	5.8%	4.5%	2.8%	2.5%

is to improve the performance of BoDeGHa by leveraging predictions from multiple repositories, we need a large collection of GitHub repositories with contributors being active in multiple repositories. Good candidate datasets are collections of repositories associated to the collaborative development of open source software packages for specific programming languages. This is because they are more likely to be stable since they are distributed through package managers, have closely related projects that increase the likelihood of the same contributor being involved in multiple repositories, and tend to avoid repositories created merely for experimental or personal reasons, or that only show sporadic traces of activity (Kalliamvakou et al., 2014).

We collected the GitHub repositories associated with the software packages that are distributed through the Cargo package manager, for the Rust programming language. In October 2021, 68,621 Rust packages were available on Cargo and 38,886 of them (i.e., 56.7%) have an associated repository on GitHub. Since we need bots to be active in the repositories to conduct our empirical study, and since bots are more likely to be present in larger and more mature repositories, we excluded packages that do not even refer to their homepage or to their documentation. This left us with 22,156 packages. Given that BoDeGHa relies on the comments made in issues and PRs to identify bot contributors, we excluded repositories having less than 100 issues or PRs. At the end of the data extraction process, the dataset contains 1,039 GitHub repositories accounting for 147,426 pairs of contributor/repository.

4.2.3 Contributor activeness in multiple repositories

As we aim to improve bot identification by leveraging predictions made on multiple repositories, we need contributors to be active in more than a single repository. This section aims to quantify how frequently contributors are active in multiple repositories. The 147,426 pairs of contributor/repository in our dataset correspond to 57,757 distinct GitHub contributors, already indicating that some contributors are active in more than one repository. Only 8,532 contributors out of these 57K (14.8%) have enough commenting activity in at least one repository for BoDeGHa to be applied. For each of these 8,532 contributors (i.e., each distinct GitHub contributor), we counted the number of repositories that each contributor was active in. Table 4.2 reports on the number and proportion of contributors in function of the number of repositories they are active in.

We observe that while most contributors (5,671 out of 8,532, 66.5%) are active in a single repository only, around one third of the contributors (2,861, i.e., 33.5%) are active in multiple repositories. We will focus on those 2,861 contributors since they correspond to those for which BoDeGHa will produce several, potentially different (i.e., *bot* and *human*) or incomplete (i.e., *unknown*) predictions. These 2,861 contributors are active in a total of 1,010 distinct repositories.

4.2.4 Number of different or incomplete predictions

We applied BoDeGHa on each of the 1,010 repositories identified in Section 4.2.3 in order to get the predictions for each of the 2,861 contributors active in two or more repositories. Under the hood, BoDeGHa downloads up to 100 PR or issue comments for each contributor active in the repository. Only the comments made during the last five years at the moment of conducting the experiment (i.e., after December 2016) are considered. BoDeGHa then analyses these comments and predicts whether the contributor corresponds to a *bot* or a *human* contributor based on several features including the repetitiveness of comments and the number of comment patterns. If a contributor has less than 10 comments, BoDeGHa classifies it as *unknown*. At the end of this process, we have a total of 41,542 predictions of which 1,146 correspond to *bot*, 10,227 to *human* and 30,169 to *unknown*. The high proportion of *unknown* predictions (73%) indicates that most contributors have less than 10 comments in the considered repositories.

Since our focus is on improving bot identification, we select contributors that were classified as *bot* at least once. Out of the initial 2,861 distinct contributors active in at least two repositories, 229 (8%) were classified as *bot* at least once. Among them, 106 (46%) were consistently classified as *bot* in all the repositories they were active in. Out of the 123 remaining contributors having been predicted as *bot* at least once, 60 have different predictions (i.e., they were also classified as *human*) and 63 have consistent but incomplete predictions (i.e., they were also classified as *unknown*).

To assess to which extent bot identification can be improved by leveraging predictions from multiple repositories, we need to determine the correct type (i.e., *bot* or *human*) of each contributor. Following the same guidelines mentioned in Section 4.1.2, two researchers (co-authors of our publication Chidambaram et al. (2022)) manually and independently determined the type of 229 contributors that were at least once predicted as *bot*. Following an inter-rater agreement process, the first step of this process ended up with an agreement on 95% of the cases. The remaining ones were discussed together, ending up with an agreement on all of them. With this process, we found that BoDeGHa incorrectly predicted *bot* in 110 cases and incorrectly predicted *human* in 31 cases. Table 4.3 summarises the number of actual bot and human contributors we found, as well as the number of *bot*, *human* and *unknown* predictions obtained for them.

4.2.5 Leveraging different predictions

Section 4.2.4 revealed that many contributors have different predictions depending on the repository BoDeGHa is applied on. In this section, we propose an approach based on the wisdom of the crowd principle to fix these different predictions.

Let WoC-P be such bot identification model. WoC-P stands for *Wisdom of the Crowd principle for Predictions* and works on top of BoDeGHa by automatically re-

Table 4.3: Number of actual bot and human contributors, and their number of *bot*, *human* and *unknown* predictions.

		predictions		
	contributors	# *bot*	# *human*	# *unknown*
actual bot	142	1,110	31	413
actual human	87	110	288	1,134
total	229	1,220	319	1,547

placing the less frequent predictions of a contributor with the most frequent ones. For example, if a contributor is predicted as *human* in 4 out of 10 repositories they contributed to and is predicted as *bot* in the remaining 6 repositories, then WoC-P gives the contributor type as *bot*. Ties are arbitrarily resolved as *human*. We applied both BoDeGHa and WoC-P on the 84 contributors that have at least two predictions of which one is *bot*. Fig. 4.4 shows, for each contributor, the number of *human* predictions, the number of *bot* predictions, and whether it is an actual bot or human. To visually distinguish overlapping points, we added a jitter of 0.25 on both axes. The diagonal line illustrates the WoC-P model: any contributor above the line will be consistently predicted as *bot* (i.e., the *human* predictions are replaced by *bot* predictions), while any contributor below will be consistently predicted as a human (i.e., the *bot* predictions are replaced by *human* predictions).

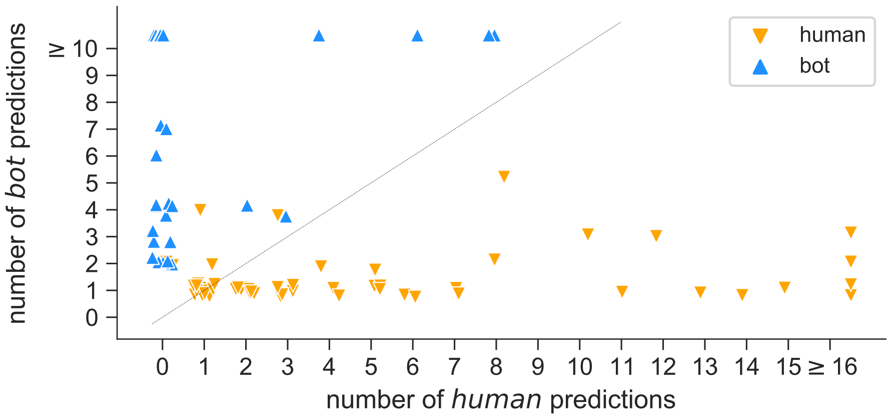


Figure 4.4: Number of *bot* and *human* predictions, each point is a contributor.

As can be observed from Fig. 4.4, the approach proposed by WoC-P seems promising. Most of the contributors mostly have predictions corresponding to their actual type. Only five human contributors have a higher number of *bot* predictions than *human* predictions. These contributors will be consistently but wrongly predicted as *bot* by WoC-P. To assess to which extent BoDeGHa can be improved by WoC-P, we evaluated both models on the 84 contributors. Table 4.4 reports on the resulting number of true positives (TP), true negatives (TN), false positives (FP), false nega-

Table 4.4: Score comparison between BoDeGHa and WoC-P.

	TP	TN	FP	FN	Acc	P	R	F1
BoDeGHa	928	288	79	31	91.7	92.2	96.8	94.4
WoC-P	959	348	19	0	**98.6**	**98.1**	**100.0**	**99.0**

tives (FN) as well as on the accuracy (Acc), precision (P), recall (R) and $F1$-scores (F1) of the two models.

We observe that WoC-P actually improves the predictions made by BoDeGHa. WoC-P replaced a total of 101 predictions out of 1,326 (i.e., 7.6%): 65 *bot* predictions were correctly converted to *human* predictions, while 36 *human* predictions were converted to *bot* predictions, among which 31 correspond to actual bots. This leads the number of false negatives to drop from 31 to 0, and the number of false positives to decrease from 79 to 19. These 19 incorrect predictions correspond to the five human contributors above the diagonal line in Fig. 4.4. As a consequence, WoC-P has higher accuracy, precision, recall and $F1$-scores compared to BoDeGHa.

4.2.6 Leveraging incomplete predictions

Till the previous section, we relied on the wisdom of the crowd principle, using the most frequent prediction to fix the less frequent predictions. This section aims to determine whether a similar approach can be followed to fix *unknown* predictions as well.

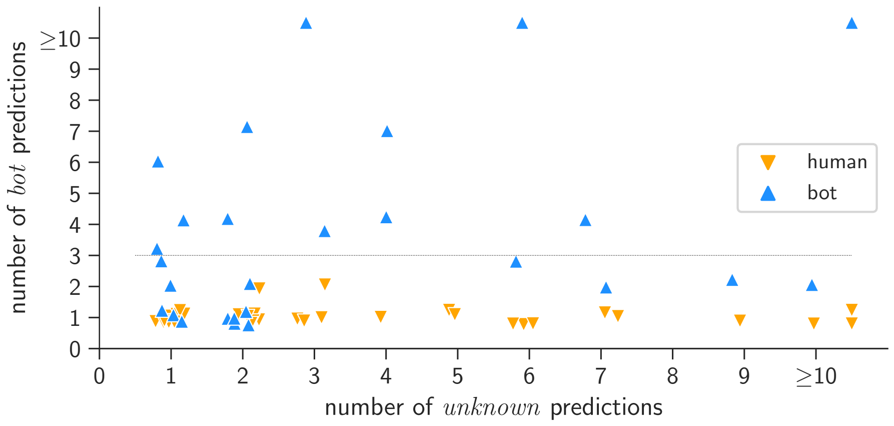


Figure 4.5: Accounts predicted as *bot*/*unknown* vs ground truth.

Fig. 4.5 shows the number of *unknown* and *bot* predictions for the 63 contributors that were either predicted *bot* or *unknown* (i.e., that have no *human* predictions). However, 33 human contributors are predicted only as *bot* or *unknown*. Converting the *unknown* predictions to *bot* for these 33 human contributors would only increase the number of incorrect predictions (FP). For instance, while converting the 184 *unknown* predictions of the 30 bots increases the number of correct predictions from 336 to 520, doing the same for the 158 *unknown* predictions of the 33 human contributors increases the number of incorrect predictions from 35 to 193.

Nevertheless, we observe that most of these human contributors have a low number of *bot* predictions compared to the actual bot contributors. For instance, there are 17 bots and no human having three or more *bot* predictions. On the other hand, all human contributors and “only” 13 bots have one or two *bot* predictions. Converting only the *unknown* predictions of contributors having three or more *bot* predictions would increase the number of correct predictions from 318 to 460, without increasing the number of incorrect predictions. However, since this threshold of “3+ *bot* predictions” is obtained by observation, it cannot be integrated into the WoC-P model without prior validation on a bigger dataset.

4.3. Summary and conclusions

Some of the existing bot identification approaches (presented in Section 3.2) consider only a limited number of activity types and their activity at repository level. This makes these approaches unable to detect a bot that is involved only in specific activity types that are not considered by them, and leading to different or incomplete predictions when used on multiple repositories. This makes it difficult to determine the actual type of contributor. This chapter addressed **Goal 1** of this thesis by developing two new bot identification models that rely on existing approaches to improve the overall performance of bot identification.

One approach leverages the predictions provided by existing bot identification approaches and provide the final decision on the type of the contributor. This resulted in an ensemble model based on Decision Tree classifier that performs better than the existing approaches. Another approach leverages the predictions provided by the bot identification tool BoDeGHa on multiple repositories and provide a final decision on the type of each contributor. This resulted in a model that uses wisdom of the crowd principle to conclude the type of contributor based on the most frequent prediction. This approach provided good improvement when used on top of BoDeGHa’s predictions.

CHAPTER 5

Differentiating bots from humans based on activities

Together with Chapter 6, this chapter aims to address **Goal 2** of this thesis. Chapter 4 presented two improved GitHub bot identification models. They are limited to specific types of activities performed by GitHub contributors, and use features based on the profile login name, comments made in issues and PRs, and git commit messages. This makes these bot identification models unable to detect bots that do not engage in such activities. But, as shown in Section 2.1 and Section 3.1, bots perform many more activity types. So, in this chapter, we assess if considering contributor activity sequences in GitHub (without restricting to a single repository) across a wider range of activity types and features based on activity patterns could potentially be useful indicators to identify GitHub bots more accurately.

The low-level, technical event types obtained through the GitHub API do not explicitly correspond to the high-level, conceptual activity types performed by a contributor in GitHub. For example, a single event type CreateEvent (as discussed in Section 2.7) can correspond to multiple activity types: creating a repository, a branch, or a tag. For this reason, Section 5.1, which is based on our publication (Chidambaram et al., 2023a), presents a detailed mapping to convert low-level events to higher-level activities, and provides a dataset of contributor activity sequences. In Section 5.2, which is based on our publication (Chidambaram et al., 2023b), we use this dataset to perform a preliminary analysis that considers a wider range of activity types to identify statistically differentiating features based on activity patterns that can identify bots.

5.1. Identifying activities from GitHub events

5.1.1 Motivation and structure

This section contributes to **Goal 2** of this thesis and is based on my co-authored publication in the International Conference on Mining Software Repositories (Chidambaram et al., 2023a). The corresponding dataset of GitHub contributor activity sequences along with the mapping from event types to activity types is publicly available on `https://doi.org/10.5281/zenodo.7740520`.

Existing bot identification approaches rely on a limited set of activity types performed by GitHub contributors, such as comments made in issues and PRs, and commit messages. Relying on GitHub's REST API `events` endpoint, which provides 17 event types, we can obtain a wider range of activities performed by contributors in GitHub. Also, by making three API queries, we can quickly obtain 300 activities that a contributor performed in the last 90 days in GitHub. However, they do not explicitly correspond to the activity types performed by a contributor. For example, a single event type CreateEvent (as discussed in Section 2.7) is used to encode different activity types: creating a repository, a branch, or a tag. Also, a single activity of closing an issue with a comment will be encoded by two events of different types: IssuesEvent and IssueCommentEvent. This raised the need to map low-level event types accessible through the `events` endpoint to higher-level activity types covering a wide range of activities such as issues, PRs, releases and repository management. We use this mapping to create a dataset of high-level activities carried out by bots and human contributors in GitHub. An important contribution of the dataset is that it encodes activities at a more conceptual level of granularity than low-level GitHub events. Another important contribution of the dataset is that it contains historical activity data that can no longer be recovered through GitHub nor its API, and that cannot be retrieved easily from third-party datasets such as GH Archive. Indeed, the size of the hourly contributor events of GH Archive ranges from 100Mb to 1000Mb, and finding all events generated over the last 105 days would result in processing more than 1.5Tb of data. Our activity dataset is suitable to perform empirical studies of how bots play a role in collaborative software development. As an illustration of this, Section 5.2 performs a preliminary analysis of activity patterns in bots and human contributors.

The remainder of the current section is structured as follows. Section 5.1.2 details the dataset construction process. Section 5.1.3 provides the data schema of the dataset and Section 5.1.4 discusses the limitations of the dataset.

5.1.2 Dataset construction

This section explains the methodology followed to extract the events carried out by GitHub contributors using the GitHub REST API and to identify higher-level activities from the events. These activities aim to facilitate the characterisation of bot and human behaviour in GitHub repositories, by enabling the analysis of activity sequences and activity patterns of bot and human contributors. Fig. 5.1 provides a

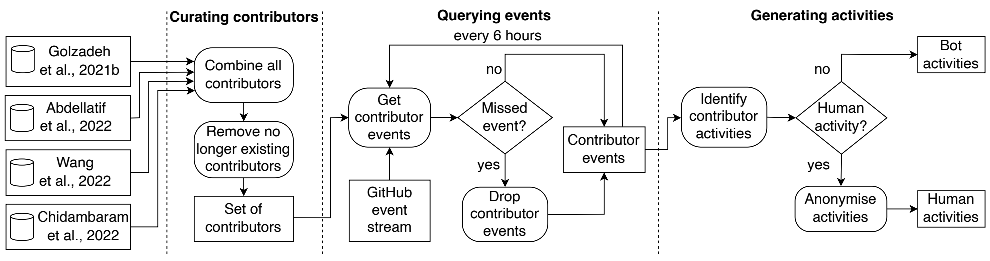


Figure 5.1: Dataset construction process.

high-level summary of this process, decomposed in three steps: (A) Curating contributors, (B) Querying events, and (C) Generating activities.

A. Curating contributors

In order to gather activities made by bots and human contributors, first we need to come up with a list of bots and human contributors. To do so, we rely on four curated datasets that were used for training the bot identification tools BoDeGHa (Golzadeh et al., 2021b) and BotHunter (Abdellatif et al., 2022) and for analysing bot usage in collaborative software development (Wang et al., 2022; Chidambaram et al., 2022). These datasets provide a list of manually verified bots and human contributors. We combined all bots identified in these ground-truth datasets and removed duplicates, leading to 890 distinct bots, and we randomly selected a similar number of human contributors.

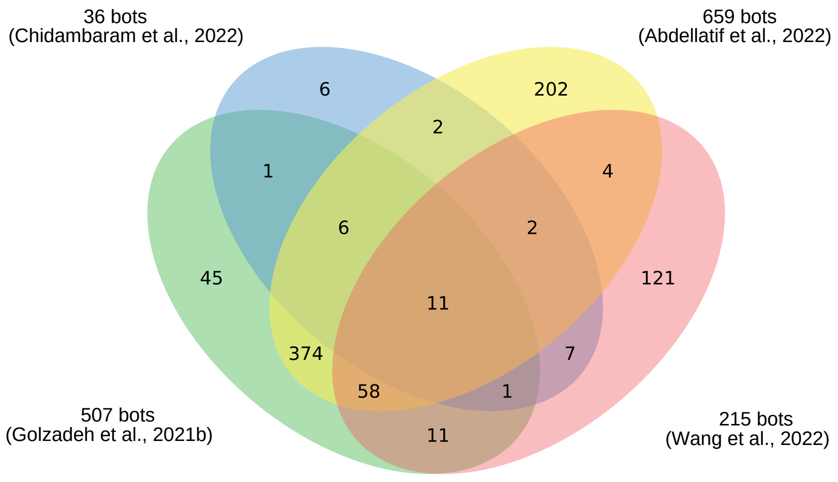


Figure 5.2: Distribution of bots across four initial datasets that will form the basis of our own dataset.

Out of the 890 considered bots, 165 are bot actors, 686 are acting through GitHub user accounts, and 39 were dismissed as they no longer exist on GitHub. Figure 5.2 shows the distribution of these 851 bots among the four curated datasets we relied on. We can observe that a majority of 507 bots are obtained from the ground-truth dataset published by Golzadeh et al. (2021b). The dataset provided by Abdellatif et al. (2022) allowed us to find 210 more bots. We added, 13 new bots identified by Chidambaram et al. (2022). The remaining 121 bots were obtained from Wang et al. (2022).

B. Querying events

Given the name of a contributor, the GitHub REST API's `events` endpoint provides all recent events that were generated by the contributor (e.g., CreateEvent when a repository is created, or IssueCommentEvent when commenting an issue). We relied on these events to generate higher-level activities. However, the `events` endpoint can only be used to retrieve up to 300 events, and only those that were generated during the last 90 days. This second limitation is not really impactful since most contributors require less than 90 days to generate 300 events. The first limitation, however, has an important implication in the case of contributors generating more than 300 events in short periods of time, since older events will no longer be available through the API.

Since our aim is to provide all the activities performed by the considered contributors during a period of 105 days, between 25 November 2022 and 9 March 2023, we needed to iteratively and frequently query the API to ensure that no event is missed. Therefore, we queried the API every 6 hours for each contributor. To ensure that no event was missed between two consecutive calls X and Y, we checked whether the oldest returned event in Y was part of the events returned in X (i.e., there was no "gap" between both event sequences). In case an event was missed, we removed the corresponding contributor from our list. We found 28 human contributors and 69 bots in such a situation. We also excluded 74 human contributors and 397 bots that did not generate any event during the considered period of time. By doing so, we retrieved 1,027,384 distinct events for 385 bot contributors and 616 human contributors.

C. Generating activities from events

Now that we have obtained the events that 385 bots and 616 human contributors performed in GitHub, the next step is to generate the activities made by these contributors based on the events they produced. To do so, we first needed to come up with a classification of high-level activities and their mapping to lower-level GitHub events. Three researchers carefully went over the documentation of GitHub REST API and the various event types to identify the high-level activities that can be deduced from them. We also manually performed various activities through the GitHub UI to observe the events generated by them in order to map events and activities. As per the documentation,[1] GitHub `events` endpoint provides 17 event types, however, we

[1]`https://docs.github.com/en/rest/using-the-rest-api/github-event-types?apiVersion=2022-11-28`

Table 5.1: Activity types and the corresponding event type(s) with their payload detail. We also indicate if the event type is required or optional.

Activity type	**Optional**	**Event type(s)**	**Payload**
Creating repository		CreateEvent	ref_type="repository"
Creating tag		CreateEvent	ref_type="tag"
Creating branch		CreateEvent	ref_type="branch"
Deleting branch		DeleteEvent	ref_type="branch"
Deleting tag		DeleteEvent	ref_type="tag"
Making repository public		PublicEvent	-
Adding repo collaborator		MemberEvent	action="added"
Forking repository		ForkEvent	-
Starring repository		WatchEvent	action="started"
Editing wiki page		GollumEvent	action="created" or action="edited"
Publishing release		ReleaseEvent	action="published"
	✓	CreateEvent	ref_type="tag"
Opening issue		IssuesEvent	action="opened"
Transferring issue		IssuesEvent	action="opened"
Closing Issue		IssuesEvent	action="closed"
	✓	IssueCommentEvent	action="created"
Reopening Issue		IssuesEvent	action="reopened"
	✓	IssueCommentEvent	action="created"
Commenting issue		IssueCommentEvent	action="created"
Opening PR		PullRequestEvent	action="opened"
Closing PR		PullRequestEvent	action="closed"
	✓	PushEvent	-
	✓	IssueCommentEvent	action="created"
Reopening PR		PullRequestEvent	action="opened"
	✓	IssueCommentEvent	action="created"
Commenting PR		IssueCommentEvent	action="created"
Commenting PR changes		PullRequestReviewCommentEvent	action="created"
	✓	PullRequestReviewEvent	action="created"
Reviewing code		PullRequestReviewEvent	action="created"
Pushing commit		PushEvent	-
Commenting commit		CommitCommentEvent	action="created"

considered only 15 as the other two event types SponsorshipEvent and PullRequestReviewThreadEvent were not performed by our considered set of contributors during the observation period and could not replicate this event. Through an iterative process, we unanimously agreed on a final list of 24 high-level activity types and their mapping to the generated 15 event types.

This mapping between activity types and event types is not one-to-one, since some activity types are obtained from sequences of two event types, and some event types may give rise to different activity types depending on the kind of data that is provided by the event type. For example, activity type *Closing issue* (with a comment) is obtained from a combination of IssuesEvent and IssueCommentEvent event type. As another example, the CreateEvent event type can correspond to one of the activity types *Creating repository*, *Creating branch* or *Creating tag*, depending on the value of its `ref_type` field. Table 5.1 lists the 24 activity types we identified as well as their mapping to event types. Optional event types are marked with a ✓in the 'Optional' column.

Using this mapping, we converted the 1M+ events from step B (Fig. 5.1) into 833,811 activities of which 649,755 are made by 385 bots and 184,056 activities by 616 human contributors. This already indicates that, on average, bots are considerably more active than humans. We provide the generated activities in two separate datasets: one for bots and one for human contributors. The latter dataset is pseudo-anonymised to comply with GDPR regulations, by hashing the names of human contributors and the repositories they are active in, and by removing all unique identifiers that could be used to reveal their identities.

5.1.3 Data schema

The activity datasets are provided as JSON files accompanied by a corresponding JSON schema. Listings 5.1 and 5.2 provide excerpts showing two activities made by a bot contributor. Along with the date and the activity type, each activity mentions

Listing 5.1: Example of a *Publishing release* activity performed by the kubevirt-bot bot account at 16:45:52 on 03-01-2023 in the *kubevirt/kubevirt* repository.

```
 1 {
 2     "date": "2023-01-03T16:45:52+00:00",
 3     "activity": "Publishing a release",
 4     "contributor": "kubevirt-bot",
 5     "repository": "kubevirt/kubevirt",
 6     "release": {
 7     "name": "v0.59.0-alpha.2",
 8     "description_length": 9834,
 9     "created_at": "2023-01-03T15:59:12+00:00",
10     "prerelease": true,
11     "new_tag": false,
12     "GH_node": "RE_kwDOBJIk984FO7NC"
13     },
14     "gitref": {
15     "type": "tag",
16     "name": "v0.59.0-alpha.2",
17     "description_length": 0
18     }
19 }
```

the name of the contributor and the repository in which the activity took place. Depending on the activity type, additional fields are provided, the details of which can be found alongside the shared dataset. For example, activity type *Publishing release* in Listing 5.1 provides additional details about the `release` (lines 7–12) and details regarding the tag associated to the release in `gitref` (lines 15–17). Similarly, activity type *Commenting issue* in Listing 5.2 provides additional details about the `comment` (lines 7 and 8), the `issue` (lines 11–17) and the `conversation` (line 20) involved in the activity. Whenever available, we provide the `GH_node` (lines 8 and 17) of the corresponding objects, a globally unique identifier to find related objects (e.g., comments, issues or PRs) on GitHub.

Listing 5.2: Example of a *Commenting issue* activity performed by the kubevirt-bot bot account at 14:13:19 on 26-11-2022 in the *kubevirt/kubevirt* repository.

```
 1 {
 2   "date": "2022-11-26T14:13:19+00:00",
 3   "activity": "Commenting issue",
 4   "contributor": "kubevirt-bot",
 5   "repository": "kubevirt/kubevirt",
 6   "comment": {
 7     "length": 255,
 8     "GH_node": "IC_kwDOBJIk985PKH4s"
 9   },
10   "issue": {
11     "id": 8294,
12     "title": "SRIOV VF interface not found in VM",
13     "created_at": "2022-08-13T11:10:06+00:00",
14     "status": "open",
15     "closed_at": null,
16     "resolved": false,
17     "GH_node": "I_kwDOBJIk985Pvz5k"
18   }
19   "conversation": {
20     "comments": 9
21   }
22 }
```

5.1.4 Limitations

A first limitation of the datasets stems in the range of activity types contained in them. We relied on GitHub's REST API `events` endpoint to identify the activities performed by contributors. However, not all activities on GitHub generate public events provided by this API endpoint. For example, locking, unlocking and labelling an issue are not provided by this API endpoint. While such data could be retrieved through the `issues` endpoint, it would result in a significant increase in the required number of API queries.

A second limitation is a consequence of the fact that the `events` endpoint returns at most 300 events performed by a contributor in the last 90 days, and that we queried the API every 6 hours. Since our goal was to provide a complete list of activities made by contributors, we had to ensure that no event was missed between consecutive calls (see Section 5.1.2). As a consequence, we had to drop all contributors that generated at least once more than 300 events in less than 6 hours. While this affected only 28 human contributors, bots are usually more active, and we had to exclude 69 of them. For example, the dependabot[bot] bot actor (corresponding to the `Dependabot` internal automation service) frequently takes less than a minute to generate 300 events. Therefore, and to a limited extent, our dataset is slightly biased towards contributors that are not "overly active".

A last limitation relates to the lack of reliability of some data provided by GitHub. For example, a PushEvent reports on the number of commits pushed through the `size` and `distinct_size` fields. However, we found that the values indicated in these fields do not always correspond to the actual number of commits that were pushed. Another example is the `merge` status reported in a PullRequestEvent that sometimes incorrectly identifies a merged PR as being unmerged (Gousios & Zaidman, 2014; Kalliamvakou et al., 2014). This last limitation does not impact any of the analysis in this thesis as we do not rely on the number of commits and the status of PRs.

5.2. Preliminary features differentiating bots and humans

This section partly addresses **Goal 2** of this thesis. It is based on my co-authored publication in the International Seminar on Advanced Techniques & Tools for Software Evolution (SATToSE) 2023 (Chidambaram et al., 2023b). We identify a series of features to quantitatively distinguish bots from human contributors based on relevant information such as their activity types, the number of repositories they are involved in, the time it takes to carry out or switch between activity types, and so on. Such a set of distinguishing features will be used in Chapter 6 to create a bot identification model to efficiently and reliably identify whether a contributor is a bot or a human based on their activities in GitHub.

The remainder of this section is structured as follows. Section 5.2.1 presents a curated dataset that will be used for analysis in this section. Section 5.2.2 details the statistical method used to compare the distribution values for bots and human contributors. Section 5.2.3 to Section 5.2.7 outline the intuition behind choosing each feature to differentiate bots from humans, and present the statistics that confirm this intuition.

5.2.1 Curated Dataset

To distinguish bots from human contributors based on their activities in GitHub, we need to have a dataset of activities that bots and human contributors performed in GitHub. For this we rely on the dataset created in Section 5.1.2. Fig. 5.3 gives the proportion of contributors and the maximal number of activities performed by contributors present in the initial dataset. We can observe some infrequent contributors that performed very few activities. Since such contributors are unlikely to be helpful in determining distinguishing features between bots and humans, we exclude them by setting a minimum threshold of 30 activities as inclusion criterion (i.e., at least roughly one activity every three days on average). The vertical dashed line in Fig. 5.3

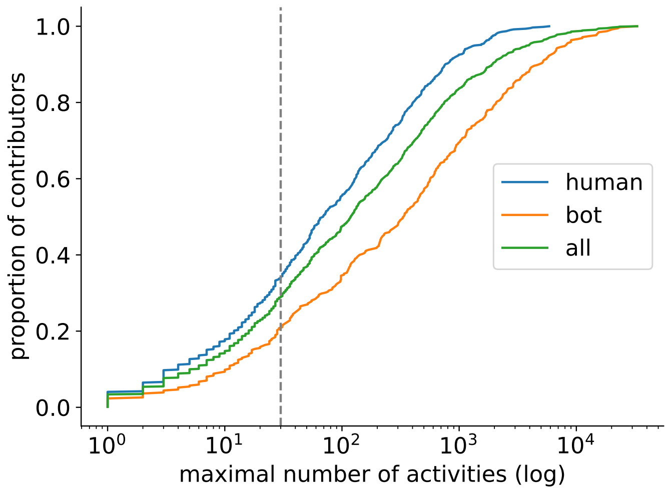


Figure 5.3: Maximal number of activities performed by the contributors (bots or humans). The vertical dashed line marks the minimum threshold of 30 activities per contributor that we used to include contributors in our analysis.

Table 5.2: Initial and curated dataset of considered contributors and activities, and top five activity types along with their number of activities in the curated dataset.

		dataset	
		initial	curated
bot	# contributors	385	305
	# activities	649,755	648,752
human	# contributors	616	408
	# activities	184,056	181,751
total	# contributors	1,001	713
	# activities	833,811	830,503

indicates this threshold. In Chapter 6, we will evaluate the performance of the bot identification model in function of the number of considered activities.

Based on this threshold of 30 activities, we excluded 80 bots and 208 humans from the dataset, together with their 3,308 corresponding activities. This left us with 305 bots and 408 humans, accounting for a total of 830,503 activities. Table 5.2 summarises the curated dataset that will be used for our analysis.

5.2.2 Statistical methods

Whenever appropriate, for the features that will be studied in upcoming sections we carry out statistical tests to compare the distribution values for bots and human contributors, using the non-parametric Mann-Whitney U test (a.k.a. Wilcoxon rank-sum test). We will reject the null hypothesis that the two distributions are equal using a significance level of $\alpha = 0.001$ after controlling for family-wise error rate using the Bonferroni-Holm method (Holm, 1979). For each test for which the null hypothesis can be rejected, we also compute the effect size using Cliff's delta (δ) (Cliff, 1993). Following the interpretation by Romano et al. (2006), we consider the effect size to be *negligible* if $\delta < 0.147$, *small* if $0.147 \leq \delta < 0.33$, *medium* if $0.33 \leq \delta < 0.474$ and *large* if $0.474 \leq \delta$.

When visualising the distributions of some metrics, we will make use of boxen plots (a.k.a. letter-value plots (Hofmann et al., 2017)) to provide a good representation of the (often skewed) distribution of the data as opposed to box plots. In a boxen plot, data representation starts at the median value and extends (i.e., draws a level line) by half of the remaining data to be covered from the current level. For example, from 50% (median), the next level towards the top will be at (50+25)%, the next level at (50+25+12.5)% and so on until 95% of the data is covered, data points after this threshold will be marked as outliers. Similarly, on the lower side, the next level will be at (50-25)%, the next at (50-25-12.5)% and so on until it reaches the point where the last 5% of the data is present, after which they marked as outliers. This facilitates the visualisation and interpretation of skewed distributions. For the same reasons we hide the outliers in these boxen plots.

Finally, a couple of features are based on the statistical dispersion of some metric. Gini (Dorfman, 1979), Theil (Theil, 1967), Kolm (Kolm, 1976), Atkinson (Atkinson, 1970) and Hoover (Hoover, 1936) are well-known econometric aggregation metrics to measure unevenly distributed data. As there seems to be a statistically significant correlation between these metrics (Vasilescu et al., 2011), choosing one over another should not impact the results of our analysis. As the well-known Gini coefficient was used in the past for a similar purpose, i.e., differentiating bots from humans (Golzadeh et al., 2021b), we will rely on it. Its value is comprised between 0 and $1 - \frac{1}{n}$ (n is the number of data points), where a value equal to 0 reflects an equal distribution while a value close to $1 - \frac{1}{n}$ expresses a maximal inequality among individuals.

5.2.3 Number of activity types

Intuition: *Bots are involved in less activity types than humans.*

We expect bots to be mostly involved in a specific set of activities. For example, a bot that keeps dependencies up-to-date will create PRs only, and is unlikely to push commits directly to the repository or to comment some issue. On the other hand, we expect human contributors to perform a wider range activities across the repositories to which they contribute. For example, human contributors close or merge incoming PRs in the repositories they maintain while they create PRs and issues in the external repositories they contribute to. To verify this hypothesis, we computed for each contributor the number of activity types performed. Fig. 5.4 shows the distribution of this number of activity types, distinguishing between bot and human contributors.

In line with our intuition, we observe a clear visual difference with a median of 3

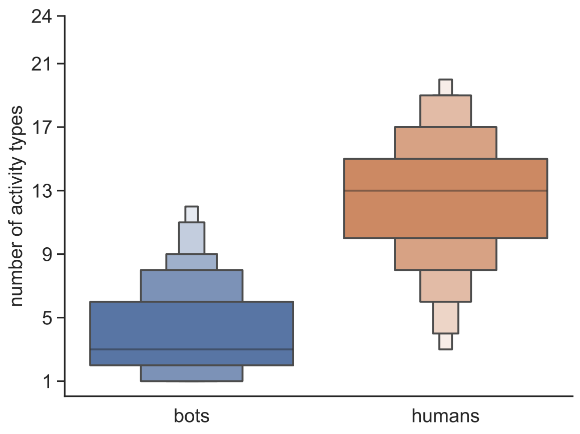


Figure 5.4: Boxen plots of the distribution of number of activity types performed by bots and human contributors.

activity types for bots compared to 13 for humans. To confirm that there is a significant difference between the number of activity types performed by bots and human contributors, we performed a Mann-Whitney U test between the two distributions. The null hypothesis is rejected ($p = 6.45e-91$) indicating that there is a statistically significant difference between the number of activity types performed by bots and human contributors in software repositories. The effect size is *large* (δ=0.88).

Conclusion: *Bots tend to be involved in less activity types than human contributors.*

5.2.4 Specialisation of activity types across repositories

Intuition: *Bots are more consistent than humans in performing their intended activity types across the repositories they contribute to.*

Not only are bots involved in less activity types than human contributors, but we expect them to be more consistent in performing these activity types across the multiple repositories to which they contribute compared to that of humans. Consider again the case of a bot keeping dependencies up-to-date. Such a bot, regardless of the repository it is deployed in, will consistently have one activity type across the repositories (i.e., creating PRs). On the other hand, depending on the repository, a human contributor may be involved in a limited number of activity types (e.g., opening an issue) or a larger number of activity types (e.g., pushing commits, closing or merging PRs, closing and commenting issues, creating releases).

To capture this intuition, we computed for each contributor the Gini coefficient

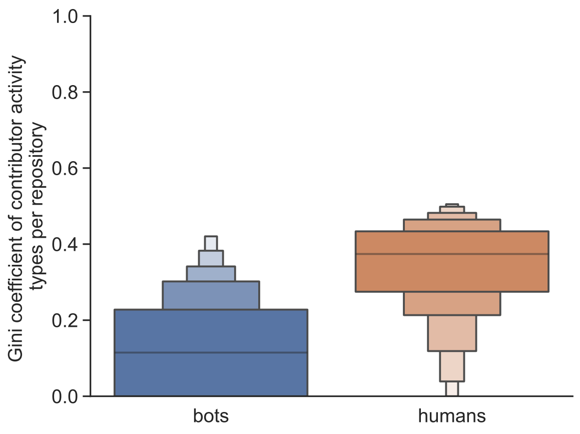


Figure 5.5: Boxen plots of the distribution of Gini coefficient for the number of activity types performed per repository by bot and human contributors.

of its number of activity types across repositories. Fig. 5.5 shows the distribution of these Gini coefficients. One can visually observe that the number of activity types performed by bots are more equally distributed (i.e., Gini is closer to 0) across repositories than for human contributors. To statistically confirm our intuition, we performed a Mann-Whitney U test between both distributions. The null hypothesis was rejected ($p = 3.12e - 68$) indicating a statistical difference between the two populations, with a *large* effect size ($\delta = 0.76$).

Conclusion: *The number of activity types performed by bots tend to be more equally distributed across repositories than the number of activity types performed by humans.*

5.2.5 Number of repositories

Intuition: *Bots are active in more repositories than humans, and tend to be used within repositories belonging to the same organisations/owners.*

We expect bots to be active in more repositories than humans as they do not have any workload restrictions. We also expect bots to be active in more repositories belonging to the same owner or organisation since it would make sense to deploy a bot in all the repositories of the same organisation or owner as long as the bot is serving its need. For humans, on the other hand, we expect to observe a more diverse behaviour, in the sense that human contributors might choose to go outside the boundaries of the repository owner or of its organisation in order to contribute to external repositories owned by another user or organisation.

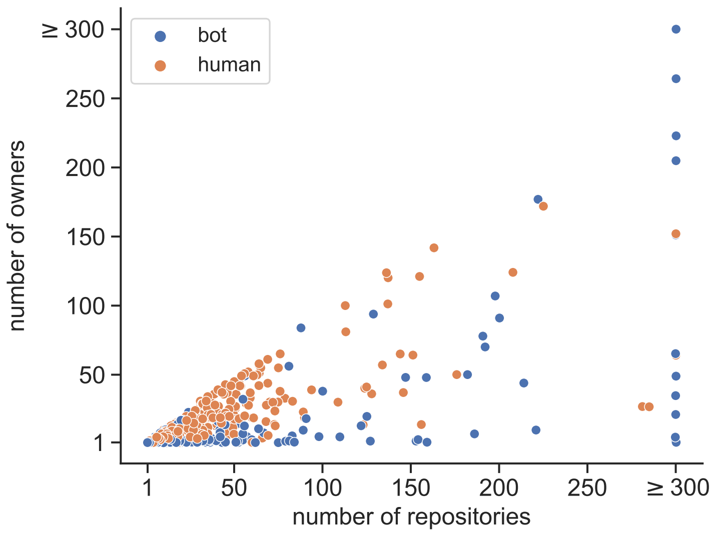


Figure 5.6: Number of repositories versus number of owners that the contributors are involved in. A jitter of 0.25 is applied on both the axes.

For each contributor, we computed the number of repositories and the number of distinct owners (or organisations) the contributor is active in. Fig. 5.6 shows a scatter plot of the number of repositories versus the number of distinct owners. To ease visualising the contributors, we added a jitter of 0.25 on both axes, and we grouped contributors that are active in more than 300 repositories or owners. Overall, 21 bots and 2 human contributors are involved in more than 300 repositories, and 10 of these bots (and no human contributor) are involved in repositories belonging to more than 300 distinct owners or organisations. On the other hand, among the 158 contributors (i.e., 22.16%) that are active in repositories belonging to a single owner or organisation, 88% are bots. Among the 81 contributors (i.e., 11.36%) active in repositories belonging to exactly two owners, 72% are bots. This suggests that bots tend to be involved in repositories belonging to a smaller number of distinct owners.

Since the number of distinct owners is upper bounded by the number of repositories, and since the number of repositories is upper bounded by the number of activities (one cannot be active in more repositories than its number of activities), and in order to capture the differences observed from Fig. 5.6, we computed for each contributor its *owner ratio* as the ratio between the number of distinct owners and the number of repositories the contributor is active in. This is, owner ratio $= \frac{\#\text{owners}}{\#\text{repositories}}$. An *owner ratio* close to 1 indicates that nearly all the repositories in which the contributor is active in belong to different owners or organisations. On the other hand, a ratio close to zero indicates that most of the repositories the contributor is active in belong to the same owner or organisation.

Fig. 5.7 shows the distribution of this owner ratio for bots and human contribu-

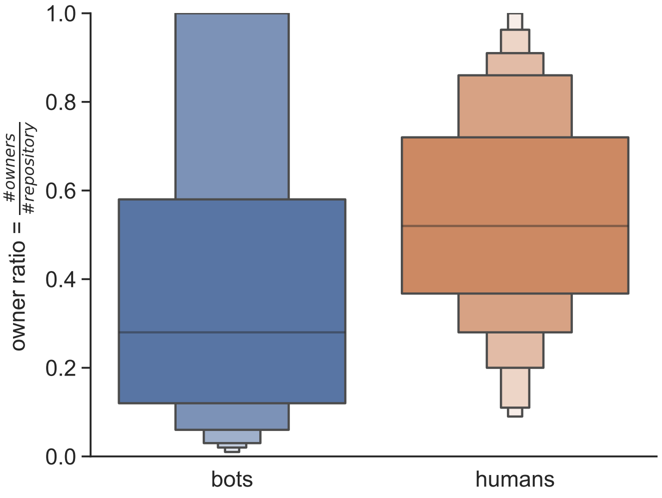


Figure 5.7: Boxen plots of the distribution of owner ratio between bots and humans.

tors. We observe that the ratio is higher for humans than for bots, indicating that human contributors tend to be involved in repositories belonging to different owners or organisations, while bots tend to be involved in repositories belonging to a more limited set of owners and organisations. We confirmed this difference between the owner ratio of bots and humans by performing a Mann-Whitney U test. The null hypothesis was rejected ($p = 9.5e - 15$) with a *medium* effect size ($\delta = 0.33$), therefore indicating a statistically significant difference between the owner ratio of bots and humans.

Conclusion: *The owner ratio is lower for bots than humans, indicating that humans tend to be active in repositories belonging to wider range of owners or organisations than bots.*

5.2.6 Time to switch between repositories

Intuition: *Bots are not suffering from context switching.*

This intuition implies that bots can easily be active in multiple repositories at the same time, while we expect human contributors to focus their workload on one repository at a time. As a consequence, we expect the time required to go from one repository to another one to be greater for human contributors than for bots.

To capture this intuition, we computed for each contributor the time to switch between two repositories. More specifically, we computed the time difference between any two consecutive activities made in two distinct repositories. Since a contributor can switch between repositories multiple times, we aggregated these time differences for each contributor by computing the median value of these differences.

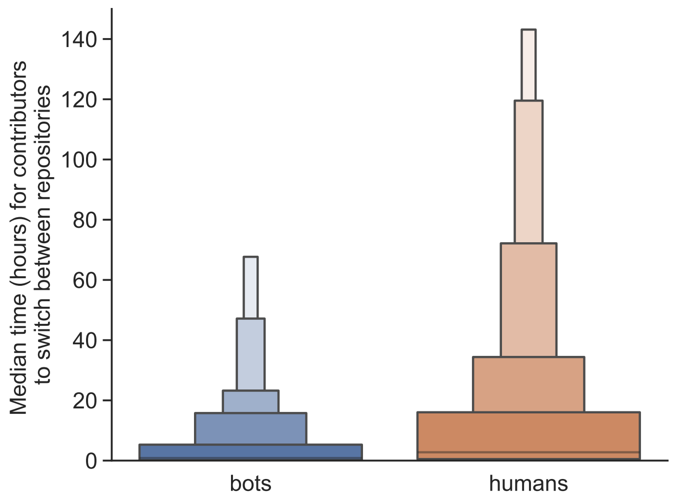


Figure 5.8: Boxen plots of the distribution of the median time (in hours) taken by bots and humans to switch between repositories.

Fig. 5.8 shows the distribution of the median time (in hours) taken by contributors to switch from one repository to another. The figure indicates that bots usually take less time to switch between repositories compared to human contributors. For instance, the median values are 0.85 hours and 2.77 hours, respectively for bots and humans. To statistically confirm this difference, we performed a Mann-Whitney U test between the two distributions. The null hypothesis is rejected ($p = 2.14e - 10$) with a *small* effect size ($\delta = 0.28$), confirming that bots take less time than human contributors to switch between repositories.

Conclusion: *Bots take less time to switch between repositories.*

5.2.7 Time between consecutive activities

Intuition: *Bots tend to have more regular work rhythms.*

Bots are not subject to the same limitations as human contributors in performing their tasks. For example, bots do not have to sleep or eat, and can afford to work at any time of the day. Their tasks are usually triggered by external events (such as a newer version of a dependency) or by activities made by other developers (e.g., a PR is submitted to the repository and a bot evaluates its code quality). Since developers can be spread around the world, a bot does not really have a fixed schedule. In contrast, humans are more likely to concentrate their work during the day (or during the evenings or the weekends if they are not professional developers). As a result, we expect the work rhythm (i.e. the time between two consecutive activities) to be much more regular for bots than for humans.

To capture this intuition, and to measure the regularity of the activities of each contributor, we computed the Gini coefficient of the time difference between consecutive activities. The lower the value, the more the contributor carries out activities on regular time intervals.

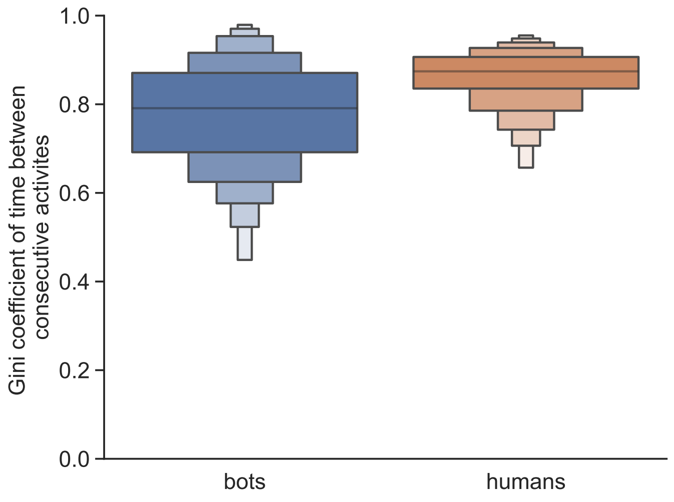


Figure 5.9: Boxen plots of the distribution of Gini coefficient of the time between consecutive activities.

Fig. 5.9 shows the distribution of these Gini coefficients, distinguishing between bot and human contributors. We observe that, as expected, the Gini coefficient of the time between consecutive activities is higher for humans than bots, indicating that humans carry out their activities on a less regular basis than bots. We performed a Mann-Whitney U test to see whether the two populations exhibit a statistically significant difference. The null hypothesis was rejected ($p = 2.14e-22$) with a *medium* effect size ($\delta = 0.42$), confirming the observed difference.

Conclusion: *Bots perform their activities on a more regular basis than human contributors.*

5.3. Summary and conclusions

In this chapter, we relied on four available ground-truth datasets and curated a set of active bots and human contributors in GitHub. This resulted in 385 bots and 616 humans. For these contributors, we queried their low-level events from the GitHub REST API's `events` endpoint and identified higher-level activities. This resulted in a dataset of 833K+ activities. Depending on this contributor activity dataset, based on our intuition and through statistical tests, we found five features that can effectively differentiate bots from human contributors in GitHub. We found that bots tend to be specialised in the sense they are involved in a smaller number of activity types than humans. We showed that bots tend to equally distribute their intended activity types across multiples repositories, and that bots are usually involved in repositories belonging to a smaller set of distinct owners or organisations. Also, we found that bots need less time to switch between the repositories they are involved in and that they tend to perform their activities more regularly than humans do.

This chapter served as a proof-of-concept to confirm that features based on activities can be used to distinguish between bots and humans, and served as a first step towards **Goal 2** of this thesis.

CHAPTER 6

An activity-based bot identification model

This chapter completes **Goal 2** of this thesis. It is based on our journal article (Chidambaram et al., 2025). Chapter 4 proposed two improved bot identification approaches, however they suffered from some of the same limitations as the bot identification approaches that they improved upon. Chapter 5 provided a ground-truth dataset of contributors in GitHub along with their activities and a set of five features that can be used as a basis to distinguish bots from humans in GitHub. This chapter proposes and evaluates an entirely new bot identification model called BIMBAS that considers a wide a range of activity types (defined in Section 5.1) and more features corresponding to the activity patterns of contributors to identify bots.

Section 6.1 provides the motivation and objectives of this chapter. Section 6.2 extends the ground-truth dataset of Section 5.1 with more bots and human contributors. Section 6.3 identifies and quantifies the performance- and efficiency-related limitations of the bot identification approaches that were presented in Section 3.2. Section 6.4 details the procedure followed to identify more features than those that were presented in Section 5.2, train and evaluate the new BIMBAS bot identification model. Finally, Section 6.5 discusses threats to validity.

6.1. Motivation and objectives

The inability to distinguish bot accounts from humans in GitHub has lead to the proposal of several bot identification approaches (Golzadeh et al., 2021b; Dey et al., 2020a; Golzadeh et al., 2020; Abdellatif et al., 2022; Liao et al., 2023). However, each approach has specific shortcomings, such as focusing on a limited subset of activities, the need for a substantial amount of API queries or data to be downloaded, the use of computationally costly features to identify bot accounts, or even the absence of a publicly available tool or script to execute the approach on recent accounts and repositories. These limitations make existing bot identification approaches difficult to use in practice for identifying large sets of contributors, highlighting the need for a bot identification tool that can be used at scale. Therefore, this chapter completes **Goal 2** of this thesis. It is based on our scientific article published in the Journal of Systems and Software (Chidambaram et al., 2025). Specifically, this chapter addresses three research objectives:

O1 *Creating a ground-truth dataset of bots and humans.* Developing a new bot detection approach requires a ground-truth dataset containing a large amount of humans and bots. The construction of a new dataset is motivated by the fact that older existing datasets are either restricted to a limited set of event types, or not sufficiently accurate. Also, creating such a dataset takes time and effort. We propose a manually curated ground-truth dataset of 2,150 contributors that were active on GitHub as of 3 May 2024. This dataset contains 1,115 humans and 1,035 bots (of which 242 are bot actors for GitHub Apps or internal automation services and 793 are bot accounts).

O2 *Identifying the limitations of existing bot identification approaches.* We found four GitHub bot identification approaches in the research literature that can be applied in practice, either because they rely on simple heuristics that are easy to implement; or because they come with a documented implementation or even a directly installable and usable tool. We study the internal workings of each approach and identify their limitations. We execute these approaches on the ground-truth dataset to compare their performance in terms of precision, recall and F1 score. We also compare their efficiency in terms of amount of data downloaded, time taken to execute and number of required GitHub API queries.

O3 *Creating a bot identification model based on activity sequences.* To overcome the limitations identified in O2 we propose BIMBAS, a novel binary classification model based on activity sequences of GitHub contributors. BIMBAS relies on a wide range of features extracted from the contributor activity sequences to accurately detect bots. BIMBAS makes use of Gradient Boosting model and achieves a performance comparable to state-of-the-art bot identification approaches.

6.2. Creating a ground-truth dataset of bots and humans

Our first research objective is to create a ground-truth dataset of active bots and humans that will serve as the basis for creating a new bot identification model (O3).

To reach objective O1, we started from three existing data sources to increase the likelihood of finding bots. We complemented this with the top contributors of several popular repositories. Since we aim to use the ground-truth dataset as a basis for a bot identification model (objective O3) based on contributor activity sequences, we impose as an inclusion filter that contributors need to have been recently active on GitHub.[1]

To ensure the accuracy of the ground-truth dataset we only include contributors that are manually checked by two raters. To do so, we applied a multi-rater labelling process to determine the contributor type. Two raters independently inspected the contributors' GitHub profiles, as well as the activities they performed on GitHub. Based on this information they labelled the contributor as either *bot* or *human* and discussed together in case of disagreement. If not enough information was available to come to a decision, the account was discarded.

As a starting point for the new ground-truth dataset we relied on the dataset presented in Section 5.1, initially containing 350 bots and 620 humans that were manually labelled. Applying the inclusion filter we retained 271 bots and 501 humans.

To further complement the ground-truth dataset, we considered the dataset used by Wyrich et al. (2021) in the context of an empirical study of the difference between bots and humans in the proportion of PRs being merged by them. They proposed a dataset of 4,654 bots, but a large majority of them (86.1%) did not pass the inclusion filter. Following the multi-rater labelling process we manually labelled the remaining 645 contributors, resulting in 506 bots and 139 humans.

Cardoen et al. (2024) provided a dataset of the commit histories of GitHub Actions workflow files in more than 32,000 repositories. The dataset contains the names of more than 62,000 contributors. Given the automated nature of GitHub Actions workflows, we expect this dataset to contain many bots. Ignoring all the contributors that were no longer active, we started by analysing all the contributors having the substring "bot" in their name. Following the multi-rater labelling process, we identified 178 bots and 45 humans. To further expand our list of bots, we applied a state-of-the-art identification tool proposed by Abdellatif et al. (2022) on randomly selected contributors. Considering only the contributors identified as bots, we followed the multi-rater labelling process until we reached 72 bots (leading to a total of 250 bots). We complemented them with 205 humans to reach an equal amount of 250 humans.

Section 1.4 reported that many bots are among the top contributors in GitHub repositories. Driven by this insight, we selected 10 popular GitHub repositories known by the authors[2] and analysed the top 30 contributors (as reported by GitHub) in each

[1]The exact definition and rationale of "active" will be provided in Section 6.4.1.

[2]Each of the considered repositories contained more than 500 watchers, more than 8k forks, and more than 25k stars.

of them. After ignoring duplicates and removing contributors who are no longer active, we followed the multi-rater labelling process and identified and included another 8 bots and 225 humans in our dataset.

Table 6.1: GitHub contributors included in the ground-truth dataset.

data sources	**total**	**#bots**	**#humans**
1. Chidambaram et al. (2023a)	772	271	501
2. Wyrich et al. (2021)	645	506	139
3. Cardoen et al. (2024)	500	250	250
4. Top contributors in 10 popular GitHub repositories	233	8	225
total	**2,150**	**1,035**	**1,115**

Table 6.1 summarises the final ground-truth dataset we obtained after following all these steps. Overall, the dataset includes 2,150 contributors, of which 1,115 humans and 1,035 bots. 791 of these bots are bot accounts, while the remaining 242 are bot actors for GitHub Apps or internal automation services. It has been made publicly available at `https://zenodo.org/records/12588134` to be used by researchers and practitioners.

6.3. Performance limitations of existing bot identification approaches

Research objective **O2** aims to compare the performance and efficiency of the most prominent approaches mentioned in Chapter 3 in order to identify their main limitations. Based on these limitations, objective **O3** (that will be presented in Section 6.4) will come up with an improved bot identification model that mitigates the identified limitations.

6.3.1 Excluded bot identification approaches

Section 3.2 provided an overview of the bot identification approaches for GitHub that have been proposed in the scientific research literature. For practical reasons, we only included the following approaches in our comparison: NBH, BoDeGHa, BoDeGiC and BotHunter.

We excluded BIMAN (Dey et al., 2020a) since it requires as input specific files that need to be obtained from the *World of Code* (Ma et al., 2021) dataset. This dataset contains the commits, blobs, trees and folders of open-source git software repositories. However, *World of Code* is not publicly available since access should be granted on an individual basis. Because of this practical limitation, we could not include BIMAN in our comparison.

We excluded BDGOA (Liao et al., 2023) since we did not find any mention of a replication package, dataset or executable tool that could be used to replicate or evaluate the proposed model. As such, we were not able to replicate the approach and could not include it as part of our comparison.

6.3.2 Experimental setup

To compare the selected bot identification approaches, we will use a *test set* corresponding to 40% of the contributors contained in the ground-truth dataset. The remaining 60% of contributors are part of the *training set* that is reserved for training the new BIMBAS model that will be proposed in **O3**. We use stratified splitting to preserve the proportion of bots and humans in the training and test set. Overall, the *training set* includes 621 bots and 669 humans, whereas the *test set* includes 414 bots and 446 humans.

Note that some approaches (BoDeGHa and BoDeGiC) work at the repository level, i.e., they require as input a repository and optionally a set of contributor names to identify which of the repository contributors are bots. By default, all contributors to the given repository will be analysed. If a contributor is active in multiple repositories, the prediction made by these approaches may depend on the repository that has been selected for analysis. For our experiment we selected the repository on which the contributor was the most active recently based on the GitHub REST API's `events` endpoint.

To evaluate the performance of the considered bot identification approaches, we rely on the usual metrics of precision (P), recall (R), F1-score (F1) and their weighted counterparts. The use of the weighted variant is motivated by the fact that our dataset is slightly imbalanced (51.9% humans and 48.1% bots).

Table 6.2 summarises the performance metrics for each considered bot identification approach. We additionally report the number of "unknown" contributors, i.e., those contributors whose type could not be determined because of intrinsic limitations of the considered approach. The presence of unknown contributors has a direct effect on the recall: a higher number of unknowns will lead to a lower recall.

Table 6.2: Performance of considered bot identification approaches on 860 contributors.

	bots				**humans**				**weighted**		
approach	P	R	F1	unknown	P	R	F1	unknown	P	R	F1
NBH	.803	.671	.732	0	.735	.848	.788	0	.768	.763	.761
BoDeGHa	.918	.512	.657	169	.861	.457	.597	223	.891	.486	.629
BoDeGiC	.812	.271	.406	278	.747	.159	.262	349	.785	.224	.346
BotHunter	.967	.928	.947	1	.937	.971	.954	0	.952	.950	.950

Table 6.3: Efficiency of considered bot identification approaches on 860 contributors.

approach	**data downloaded**	**time**	**API queries**
NBH	-	0.01 sec	-
BoDeGHa	3.83 GB	7.7 h	10,222
BoDeGiC	23.3 GB	23.1 h	-
BotHunter	0.261 GB	20.8 h	37,240

Table 6.3 reports on the efficiency of the considered approaches in terms of amount of data downloaded and execution time. Each approach was executed on the same

system with an Intel Xeon W-1290P 3.7 GHz CPU processor running Fedora 34 (Server Edition). The downloaded data is measured using the network monitoring tool NetHogs (version 0.8.7). We also count, for those approaches relying on GitHub, the number of REST API queries used. Such information is quite relevant, since GitHub imposes an API rate limit of maximum 5,000 queries per hour. Exceeding this limit results in a waiting time until the query limit is reset by GitHub. For the approaches that use the GitHub REST API, we report on this waiting time.

6.3.3 Name-Based Heuristic

The name-based heuristic (NBH) naively identifies a contributor as *bot* if the substring "bot" appears somewhere in its name. However, in Section 4.1.3 (Table 4.1 on page 45), we executed this approach on 540 contributors belonging to 27 repositories and obtained a recall of $R = 0.520$ for detecting bots. This reveals an important limitation of this heuristic, namely that it yields many false negatives, i.e., bots that do not contain the substring "bot" in their name (e.g., bors, micronaut-build, id-jenkins, strapi-cla). Conversely, this heuristic leads to false positives when humans have the string "bot" as part of their name. For example, the last names Cabot and Abbott are not uncommon for humans. For instance, in our dataset presented in Section 6.2, NBH falsely identified 68 humans as bot due to the presence of "bot" in their name, leading to a precision of $P = 0.803$ for bots. Line NBH of Table 6.2 summarises the performance results, with an overall weighted precision $P = 0.768$ and recall $R = 0.763$, confirming the presence of many false positives and false negatives.

From Table 6.2, one can observe that the recall of $R = 0.671$ for bots is higher than the recall of $R = 0.520$ that was observed in Table 4.1. This is because the recall for NBH depends only on the proportion of bots that have "bot" in their name. So, the recall reported for this approach in Table 6.2 is an overestimation since, by construction, our dataset (in Section 6.2) contains a higher proportion of such bots.

NBH is extremely efficient in time and memory. Since it only relies on contributor names, it does not require any other data to be downloaded or any API queries to be executed, implying that bots can be identified almost instantly.

6.3.4 BoDeGHa

As presented in Section 3.2.1, Golzadeh et al. (2021b) developed BoDeGHa, a bot identification tool that relies on the repetitiveness of contributors comments to identify bots. A limitation of BoDeGHa is that it restricts itself to issue and PR comments, making it unable to detect bots that do not engage in such activities. Even when a contributor is engaged in such activities, BoDeGHa requires at least 10 comments (by default) to provide a prediction. This explains why BoDeGHa could not give a prediction for 392 contributors (169 bots and 223 humans) during its execution on the test set (see Table 6.2).

While evaluating BoDeGHa, we identified some overly restrictive condition in its source code to avoid exceeding the API rate limit. We modified the code to relax this condition in order to optimise BoDeGHa's execution and created a PR that has been merged into BoDeGHa's GitHub repository.[3] We executed this improved version of BoDeGHa with its default parameters on the contributors of the test set. It took 7.7 hours (464 minutes), required 10,222 API queries and downloaded 3.83 GB of data to provide its predictions. A reason for BoDeGHa's high execution time is that it depends on the combination of features that are computationally costly (calculating and comparing Levenshtein and Jaccard distanceof text fragments). Overall, BoDeGHa achieved a weighted precision of $P = 0.891$. Since BoDeGHa could make a prediction only for 468 contributors, it achieved a very low recall of $R = 0.486$.

6.3.5 BoDeGiC

In a follow-up work, Golzadeh et al. (2020) developed BoDeGiC an alternative to BoDeGHa as discussed in Section 3.2.1. BoDeGiC uses an approach that is similar to the one of BoDeGHa, but applied to git commit messages rather than to issue and PR comments. Unlike BoDeGHa, BoDeGiC works directly with a given (local) git repository. It does not take a GitHub repository as input and does not need to use the GitHub API.

Since cloning all the repositories is time-consuming and requires downloading a large amount of data, and since BoDeGiC only requires the commit messages, we decided not to download the git blobs, which are used to store a file's binary data, along with the size of that data. To do so, we adapted the `git` command used to clone repositories to exclude blobs (`git clone -filter=blob:none -no-checkout <repo>`). With this command, for example, the *servo/servo* repository on GitHub took only 10.5 seconds and 114MB to be cloned while cloning this repository with all blobs would have taken 128 seconds and 1.14GB. The process of cloning all considered repositories took one hour while predicting the type of contributor took 22.1 hours, so in total it took 23.1 hours to execute and required 23.3GB of data.

BoDeGiC was able to provide a prediction for 233 contributors. It could not provide a prediction for the remaining 627 contributors (278 bots and 349 humans) because they do not reach the minimum 10 commits required by BoDeGiC to make a prediction. Although, BoDeGiC achieved an overall weighted precision of $P = 0.785$, it has very low recall of $R = 0.224$ due to the latter reason.

6.3.6 BotHunter

As described in Section 3.2.1, Abdellatif et al. (2022) proposed BotHunter, a Python script that executes a bot identification model based on a Random Forest classifier. Also, as we saw in Section 3.2.2, BotHunter uses variants of NBH as part of its features.

While evaluating BotHunter, we discovered that it retrieves only 30 events per API call while the GitHub API allows to retrieve up to 100 events at once. We therefore adapted the source code of BotHunter to retrieve up to 100 events per API call, thus reducing the number of API queries and reducing its execution time since the hourly API rate limit is reached less frequently. We created a PR of these code

[3] `https://github.com/mehdigolzadeh/BoDeGHa/pull/25`

modifications to the GitHub repository of BotHunter[4] which was accepted by the repository maintainer and is now integrated in the latest release of BotHunter.

We applied this modified version of BotHunter on the contributors of the test set. With the notable exception of a contributor that no longer exists on GitHub, BotHunter was able to provide predictions for all the other contributors. Table 6.2 shows that BotHunter exhibits the best performance, reaching a precision of $P = 0.952$ and recall of $R = 0.950$. On the other hand, Table 6.3 shows that BotHunter consumed 37,240 queries, downloaded 261 MB data and took 20.8 hours to provide the predictions. This long execution time includes the 50 minutes of waiting time due to the fact that BotHunter exceeded the GitHub API rate limit seven times.

6.3.7 Summary

Many of the considered approaches require, or specifically target, contributors to be involved (at least) in specific activity types (e.g., committing for BoDeGiC or commenting for BoDeGHa), making them unable to accurately determine the type of the contributors that are not involved in these activities. To some extent, deciding which approach should be used is basically a trade-off between efficiency (at scale) and performance.

6.4. Creating the BIMBAS bot identification model

We identified the performance limitations of the existing bot identification approaches in Section 6.3. The goal of the current section is to develop a new model that identifies whether a contributor is a *bot* or a *human* that: (i) exhibits a performance comparable to the state-of-the-art; (ii) can be used to predict contributors that are involved in other (or more) activity types than the usual commit-, issue- or PR-related activities; (iii) requires a low amount of data to be downloaded; and (iv) can efficiently classify thousands of contributors in limited amount of time. The current section focuses on requirements (i) and (ii), while (iii) and (iv) will be addressed in Section 7.2.

The current section is structured as follows. Section 6.4.1 explains how to construct activity sequences based on the events provided by GitHub's REST API `events` endpoint. Section 6.4.2 details the features we compute from the activity sequences and presents their rationale. Section 6.4.3 explains the procedure that we follow to impute missing values, select the best classification model, and eliminate unimportant features. Based on the output of this process, we propose BIMBAS, a classification model distinguishing bots and humans based on their activities. Section 6.4.4 evaluates the performance of BIMBAS on the test set. Section 6.4.5 discusses the most important features contributing to the predictions made by BIMBAS. Section 6.4.6 discusses the misclassified cases. Finally, Section 6.6 summarises the resulting model.

The replication package containing the material for creating and evaluating BIMBAS is available online.[5]

[4]`https://github.com/ahmad-abdellatif/BotHunter/pull/5`

[5]`https://github.com/natarajan-chidambaram/BIMBAS_RABBIT_replication_package`

6.4.1 Extracting activity sequences

In order to develop a new bot identification tool that will not require a lot of data and can consider a wide range of activity types to give a prediction for many contributors, we propose a novel approach that is entirely based on activity sequences.

At the moment of this study, the `events` endpoint allowed us to retrieve up to the last 300 public events that were generated by a contributor during the last 90 days. These low-level events will then be converted to fine-grained activity sequences using the mapping provided in Section 5.1 (Table 5.1 on page 59). Relying on activity sequences has two main benefits: (i) events can be retrieved from the `events` endpoint using at most three API queries, implying the hourly API rate limit will not be reached before around 1,666 contributors; and (ii) the activities that can be obtained from these events cover a wide range of all possible activities a contributor can do on GitHub, implying that we will be able to categorise more contributors, even those not active in committing or commenting. This section explains the procedure that we followed to retrieve the recent events from the `events` endpoint for all the contributors present in our dataset and convert them to activities sequences.

For each contributor, we queried the `events` endpoint, as of 3 May 2024, and retrieved 376,638 events performed by 2,150 contributors (194,863 by 1,035 bots and 181,775 by 1,115 humans). In this chapter, we ignored all GitHub contributors that performed less than five GitHub events, as they would not provide enough information for a bot identification model to make any conclusive decision. Converting the low-level event types to higher-level activity types following the mapping presented in Table 5.1 resulted in a total of 337,246 activities performed by the 2,150 contributors of the ground-truth dataset. 182,218 of these activities were performed by 1,035 bots, while 155,028 activities were performed by 1,115 humans.

6.4.2 Selecting features

Previous research studies (Kazi Amit et al., 2023; Zhang et al., 2022) showed that bots are significantly faster than humans in posting the first PR comment as well as in responding to a PR comment. Also, in Section 5.2, we observed many differences between the activities made by bots and those made by humans. Based on these differences we suggested five behavioural features to capture the differences between bots and humans: the number of activity types, the inequality of number of activity types across repositories, and the inequality of time between consecutive activities. In addition to this, we observed a significant difference in the number of repositories contributed to by bots and humans, and in the median time for them to switch between repositories.

We took these five features as an initial set of features. We extended this feature set by considering a wide range of counting metrics related to contributors (e.g., their number of activities, number of activity types, and number of repositories contributed to) as well as temporal metrics related to their activity sequences (e.g., duration between consecutive activities in a repository, and time difference between consecutive activities of different activity types). Several of these metrics are computed at the level of a single repository (e.g., NTR) or a single activity type (e.g., NAT) and therefore

Table 6.4: List of counting metrics considered and the intuition behind them.

Acronym	Feature	Intuition
NA	Number of Activities	Since bots are automated agents, we expect them to produce more activities than humans as they do not suffer from the same limitations as humans (e.g., the need for sleep).
NT	Number of (activity) Types	Bots are expected to perform a smaller range of activity types than humans since they are likely to be specialised towards specific tasks.
NR	Number of Repositories contributed to	Bots are expected be involved in more repositories than humans.
NOR	Number of Owners of Repositories contributed to	Many bots are expected to be used by a small number of repository owners, as they may have been developed or configured by those owners specifically for their repositories. Humans on the other hand, have the freedom to decide which repositories they contribute, regardless of who owns the repository.
ORR	Owner-Repo Ratio = $\frac{\#\text{NOR}}{\#\text{NR}}$	We expect many bots to be used by small number of repository owners that use these bots in most of their repositories. In contrast, human accounts can be freely involved in multiple repositories belonging to a wide range of repository owners.

have to be aggregated. Since we do not know in advance which aggregation functions will be the most useful for the model, we aggregate them using mean and median for central tendency, std (standard deviation) and IQR (interquartile range) for dispersion, and Gini for inequality. This leads us to a total of 45 features, of which 5 are non-aggregated and 8*5 are aggregated features. The entire list of considered features is reported in Table 6.4 and Table 6.5, together with the rationale for selecting these features.

6.4.3 Model selection

In this section we explain the process we followed to come up with a classification model for detecting bots. More specifically, Section 6.4.3 explains how we handle and impute missing values for the features we selected. Section 6.4.3 details the approach we followed to identify the best classification model (classifier and its hyperparameters). Section 6.4.3 explains the methodology followed to remove the features that do not contribute to the predictions made by the classification model.

Handling and imputing missing values

Some machine learning classifiers do not support the presence of missing values. Since the values of several features cannot be computed for contributors that are exclusively working in a single repository (e.g., DAAR) or exclusively performing a single activity type (e.g., DCAT), we apply a two-step process to impute such missing values and make the model aware of this. More specifically, we (1) replace missing values with the median value of the corresponding feature, and (2) add a Boolean indicator to signal

Table 6.5: List of aggregated metrics considered and the intuition behind them. Each of the metrics below is aggregated using five aggregation functions: mean, std, median, IQR, and Gini.

Acronym	Feature	Intuition
NAR	Number of Activities per Repository	The number of activities per repository might be high for bots as they are intended to perform repetitive tasks and can work continually without needing breaks.
NAT	Number of Activities per Type	The number of activities per activity type might be higher for bots as they are specialised in performing specific activity types in repositories.
NCAR	Number of Consecutive Activities in a Repository	Bots do not suffer from context switching, hence they are expected to switch more easily and more frequently between different repositories.
NTR	Number of (activity) Types per Repository	We expect bots to be specialised in the activities they do within repositories, hence the number of activity types they have across repositories is more likely to be constant.
DCAR	Duration of Consecutive Activities in a Repository	If bots tend to switch between repositories, the time spent in a repository for carrying out consecutive activities might be lower than for humans.
DAAR	Duration between Activities Across Repositories	Since bots do not suffer from context switching, we expect them to take less time to have activities in multiple repositories than humans.
DCA	time Difference between Consecutive Activities	Since bots are automated scripts, they may be very fast in carrying out their next activity after the previous one. The same bot might even work in parallel in multiple, not necessarily related, repositories.
DCAT	time Difference between Consecutive Activities of different Types (in hours)	Since bots do not suffer from context switching, we expect them to switch more swiftly between activities of different types.

to the model whether a missing value was imputed (van Buuren, 2012). Combining the imputed value and the Boolean indicator allows improving model performance (Van Ness et al., 2023). This two-step process is part of the model pipeline, i.e., the missing values are computed based on training data only, to avoid the model becoming contaminated by unseen data. Fig. 6.1 depicts this generic pipeline, the "Estimator" step abstracting the actual model in use.

Selecting a classification model

To differentiate bots from humans based on their activity sequences, we rely on a classification model based on a binary classifier since we only have two classes (*bot* and *human*).

We selected seven different classifier types that are commonly used and have the ability to perform binary classification: Decision Tree (Safavian & Landgrebe, 1991), Random Forest (Breiman, 2001), Support Vector Machines (Brereton & Lloyd, 2010), Gradient Boosting (Friedman, 2001), XGBoost (Chen & Guestrin, 2016), Linear Discriminant Analysis (Hastie et al., 2001), and Gaussian Naive Bayes (Chan et al., 1982).

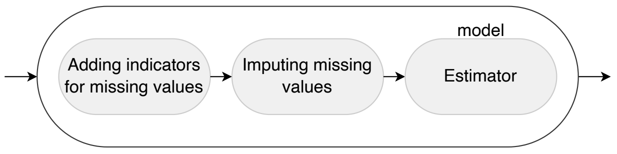


Figure 6.1: Generic pipeline for training and evaluating the classification model.

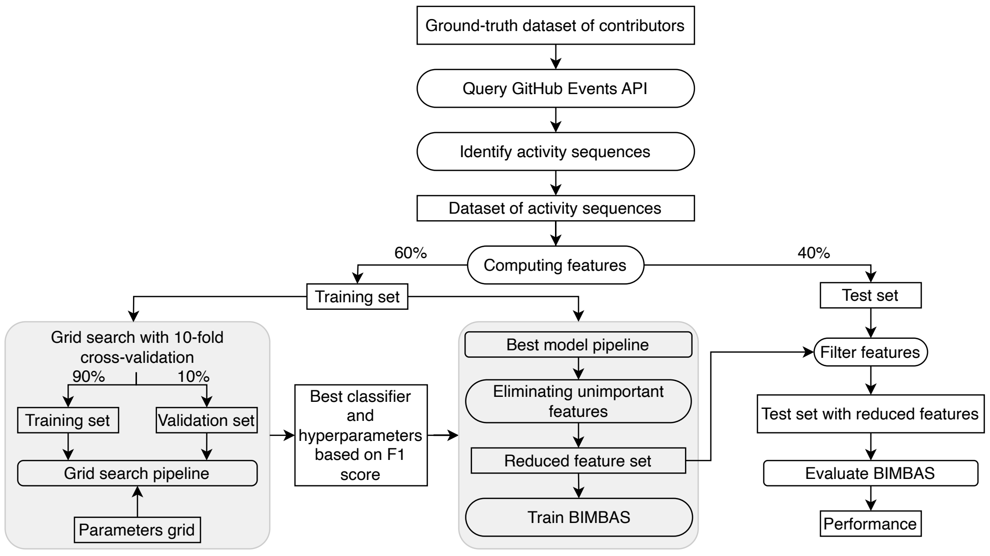

Ground-truth dataset of contributors
Query GitHub Events API
Identify activity sequences
Dataset of activity sequences
60%
Computing features
40%
Training set
Test set
Grid search with 10-fold cross-validation
90%
10%
Training set
Validation set
Grid search pipeline
Parameters grid
Best classifier and hyperparameters based on F1 score
Best model pipeline
Eliminating unimportant features
Reduced feature set
Train BIMBAS
Filter features
Test set with reduced features
Evaluate BIMBAS
Performance

Figure 6.2: Overview of the process followed to select the best model, eliminate features and to evaluate BIMBAS.

All the considered classifier types accept a set of hyperparameters that can be tuned to increase the model performance. We followed a grid-search 10-fold cross-validation hyperparameter tuning process (Witten & Frank, 2002) to find, for each classifier type, the values that should be used for these hyperparameters. In total, we consider 13,021 combinations of classifier types and hyperparameter values. The process is illustrated in Fig. 6.2 (leftmost gray box).

To train and evaluate these 13K+ models, we used the features corresponding to activity sequences of each contributor present in the training set. The training set contains 669 (out of 1,115) humans and 621 (out of 1,035) bots, i.e., 60% of all the contributors. We followed a 10-fold cross-validation process to have an estimate on the model's performance on unseen data. We relied on a stratified shuffle split strategy to maintain similar proportions of humans and bots within each fold. We measured the resulting performance of each model based on the usual performance measures of weighted precision (P), recall (R), F1 score, and area under the ROC curve (AUC). As the dataset is almost balanced, we used the ROC curve to evaluate and obtain a good overview of the model's performance across different probability thresholds.

Table 6.6 reports on the results of this process. To allow us to interpret and compare the performance of the models, we also included NBH as a baseline. Since we cannot list all the considered combinations, we report for each type of classifier on the results obtained by the best combination of hyperparameters in terms of weighted F1 score, as this score reflects the precision and recall of a model through a single, easy to compare value. Among all considered combinations, Gradient Boosting is the best overall performer, with the highest F1 score (for bots, humans, and weighted overall), the highest precision for bots, and the highest AUC score.

Table 6.6: Performance of the best model for each considered classifier type, in descending weighted F1 score.

	bots			**humans**			**weighted**			
classifier type	P	R	F1	P	R	F1	P	R	F1	AUC
Gradient Boosting	**.923**	.939	**.930**	.944	**.925**	**.934**	**.934**	**.932**	**.932**	**.970**
Random Forest	.897	**.947**	.921	**.948**	.899	.922	.924	.922	.922	.969
XGBoost	.900	.935	.917	.938	.903	.920	.920	.919	.919	.967
Decision Tree	.891	.931	.909	.933	.891	.911	.913	.910	.910	.924
Linear Discriminant Analysis	.874	.916	.893	.921	.876	.897	.898	.895	.895	.961
Support Vector Machines	.820	.806	.812	.823	.833	.827	.822	.820	.820	.833
Gaussian Naive Bayes	.865	.597	.705	.710	.912	.798	.785	.760	.753	.893
NBH (baseline model)	.765	.669	.714	.725	.810	.765	.744	.742	.740	.739

Eliminating features

We trained and evaluated the models on a set of 45 initial features. However, not all these features have the same importance during the classification process, and some of them may be redundant or may contribute little or not at all to the decision of the model. To remove unimportant features, we rely on the well-known *recursive feature elimination* (RFE) technique (Guyon et al., 2002). RFE aims to identify and eliminate the least important features (i.e., those that do not contribute much to the model performance) by recursively considering smaller and smaller sets of features. We chose the best performing Gradient Boosting binary classification model and applied RFE in a 10-fold cross-validation loop on the training set. At the end of the process, RFE identified the following seven features that can be removed without any compromise on the model performance: NR, $\mathsf{DCA}_{\mathsf{IQR}}$, $\mathsf{NAR}_{\mathsf{std}}$, $\mathsf{NTR}_{\mathsf{IQR}}$, $\mathsf{NCAR}_{\mathsf{median}}$, $\mathsf{NCAR}_{\mathsf{Gini}}$, $\mathsf{DCAR}_{\mathsf{Gini}}$.

6.4.4 Training and evaluating BIMBAS

The grid search cross-validation explained in Section 6.4.3 allowed us to identify the best classifier type (Gradient Boosting) and its corresponding hyperparameters. In Section 6.4.3, we identified that 38 features are important for the model to exhibit good performance. In this section, we introduce, train and evaluate BIMBAS, a "Bot

Identification Model Based on Activity Sequences". BIMBAS implements the selected classifier and its hyperparameters, and takes as input the 38 identified features.

Table 6.7: Performance comparison of BIMBAS against existing approaches on the test set of 860 unseen contributors.

	bots			**humans**			**weighted**		
approaches	P	R	F1	P	R	F1	P	R	F1
NBH	.803	.671	.732	.735	.848	.788	.768	.763	.761
BoDeGHa	.918	.512	.657	.861	.457	.597	.891	.486	.629
BoDeGiC	.812	.271	.406	.747	.159	.262	.785	.224	.346
BotHunter	.967	.928	.947	.937	.971	.954	.952	.950	.950
BotHunter without NBH	.984	.587	.735	.722	.991	.836	.848	.797	.787
BIMBAS	.883	.911	.897	.915	.888	.901	.899	.899	.899

BIMBAS is trained on the full training set (60% of all contributors). To assess its performance on unseen data, and therefore to validate BIMBAS, we applied it on the test set containing the remaining 40% contributors (i.e., 414 bots and 446 humans). Table 6.7 reports on the results of BIMBAS on this test set. Since the test set is the same than the one we used in Section 6.3, we also report on the results obtained by the other bot detection approaches for comparison.

BIMBAS correctly identified most of the bots (377 out of 414) with a precision $P = 0.883$ and a recall $R = 0.911$. Similarly, most of the humans (396 out of 446) that are present in the test set were correctly identified by BIMBAS with a precision $P = 0.915$ and recall $R = 0.888$. Overall, BIMBAS reaches a precision $P = 0.899$ and a recall $R = 0.899$, making it the second most performant bot detection approach.

However, in Section 6.3.3 we mentioned that the good performance of NBH could be explained by the high proportion of bots in the ground-truth dataset that have "bot" in their name. The most important features of BotHunter (e.g., *account name* and *account login*) are variations of NBH, relying on the presence of "bot" or "automate" (Abdellatif et al., 2022). This is likely the reason why BotHunter correctly identified 97.7% of the bots that have "bot" in their name, whereas it identified only 66.7% of the bots that do not. To get a better understanding of the actual performance of BotHunter on less obvious cases of bot contributors, we executed a modified version of BotHunter that no longer relies on the presence of "bot" or "automate" to make its decision.[6] The results are provided in Table 6.7 as "BotHunter without NBH". As anticipated, this variant of BotHunter has more difficulties to identify bots, with a decrease of recall for bots from $R = 0.928$ to $R = 0.587$, a decrease in overall recall from $R = 0.950$ to $R = 0.797$ and a decrease in overall precision from $P = 0.952$ to $P = 0.848$. This confirms that the performance of BotHunter heavily depends on the NBH-related features to detect bots and is less effective in detecting non-obvious cases of bots. On the other hand, BIMBAS makes no distinction between contributors based on their names, and bases its decision exclusively on the activity sequences. As such, its performance does not depend on the presence of "bot" in the name, nor of any other substring.

[6]It did not suffice to simply change the names of those contributors, since BotHunter requires the

6.4.5 Feature importance

In order to identify the most contributing features (among the 38 remaining ones), we applied the *permutation importance* (Breiman, 2001) technique on the test set. This model inspection technique measures the importance of each feature by randomly shuffling the values of one feature at a time, and measuring how this affects the performance of the model. We applied this technique in a 10-fold cross-validation setting on the test set, meaning that each feature is shuffled 10 times and the resulting F1 scores are aggregated.

Table 6.8: Top five important features for distinguishing bots from humans in test set, reporting their median values and effect size.

		median		**effect size**	
acronym	**feature**	bots	humans	δ	interpretation
NT	number of activity types	4.0	10.0	.725	large
NOR	number of owners of repositories contributed to	1.0	4.0	.554	large
$\mathsf{DCAT}_{\mathsf{median}}$	median time difference between consecutive activities of different types	0.003 h	0.125 h	.482	large
$\mathsf{NAT}_{\mathsf{median}}$	median number of activities per type	29.5	6.0	.676	large
$\mathsf{NAT}_{\mathsf{mean}}$	mean number of activities per type	45.0	14.9	.703	large

Table 6.8 reports on the 5 most important features, i.e., on the 5 features that have the higher impact in terms of F1 score when shuffled. This table also reports on the median value of these features, distinguishing between bots and humans. To determine whether there is a statistically significant difference between bots and humans for these features, we performed Mann-Whitney U tests (Mann & Whitney, 1940). The null hypothesis, stating there is no difference between the two populations, was consistently rejected with a significance level $\alpha = 0.001$ after controlling for family-wise error rate with the Bonferroni-Holm method (Holm, 1979). The effect size of these tests, based on Cliff's δ (Cliff, 1993), reveals a *large* difference for these features between bots and humans.

6.4.6 Analysing the misclassifications

The evaluation of BIMBAS revealed that is performing well on the test set. Nevertheless, BIMBAS misclassified 37 out of 414 bots as humans (FN) and 50 out of 446 humans as bots (FP). We manually tried to find possible reasons for these misclassifications.

A first observation was that many misclassified contributors performed very few activities. For example, eclipse-metro-bot and arduino-ci-script-bot are bots that respectively have only 6 and 8 activities. This lack of data makes it difficult for BIMBAS to determine if their behaviour is closer to humans or bots. To quantify the impact of the number of activities on misclassifications, we measured the proportion of misclassified contributors in function of the number of withheld activities for each contributor.

exact contributor name to retrieve its data.

Fig. 6.3 shows this proportion, revealing that a lower number of activities coincides with a higher proportion of misclassified contributors. For instance, when applied on the latest eight activities of each contributor, 25% of the contributors are misclassified. When applied on the latest 25 activities, 16% of the contributors are misclassified. In contrast, the proportion of misclassified contributors does not exceed 10% starting from 107 activities.

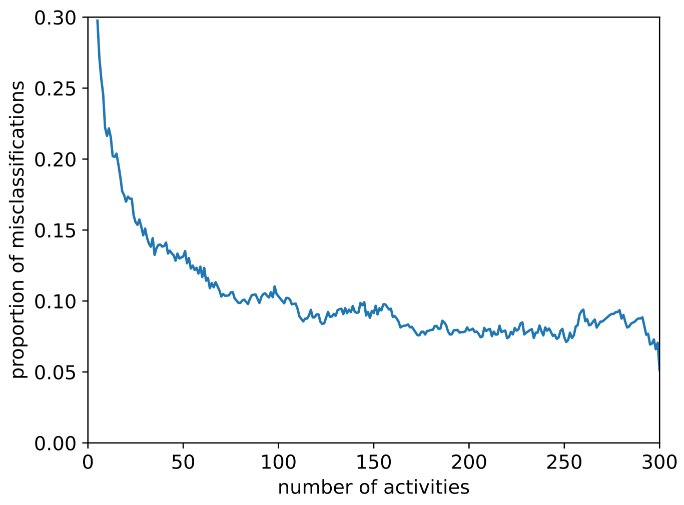


Figure 6.3: Proportion of misclassified contributors in function of number of considered activities.

Another reason for misclassified bots is that they are taking a long time to switch between activity types ($\mathsf{DCAT}_{\mathsf{median}}$), although they are involved in multiple activity types (NT) and the mean and median number of activities per activity type ($\mathsf{NAT}_{\mathsf{median}}$ and $\mathsf{NAT}_{\mathsf{mean}}$) is similar to that of humans. For example, bot-gradle is a bot that performed 244 activities belonging to 9 different activity types, but its $\mathsf{DCAT}_{\mathsf{median}}$ is 0.307 whereas its $\mathsf{NAT}_{\mathsf{median}}$ and $\mathsf{NAT}_{\mathsf{mean}}$ are 7 and 21.375 respectively. Comparing these values with the corresponding median for bots and humans in Table 6.8 conveys that it is difficult for BIMBAS to classify them as bots.

We also found instances of misclassified bots whose main or only purpose is to mirror (e.g., migrate, copy-paste or translate) human activities coming from other sources (e.g., another repository or another bug tracking system). For example, zx2c4-bot is a mirroring bot that creates and deletes tags and branches, closes PRs, and pushes commits to GitHub that were made on an external git repository. Such bots are difficult to distinguish from humans, given that each of their mirrored activities actually originate from some human activity.

Another reason for misclassifying humans is that they may be involved in very few activity types. One of the misclassified humans performed 300 activities belonging to a single activity type (pushing commits). While it is expected to have humans involved in this activity type, it is unlikely for them to be involved in only a single activity type, since the median number of activity types for humans is 10 (see NT in Table 6.8). Indeed, the test set included only one human contributor that was exclusively pushing commits, so it can be considered to be an outlier in the class of humans. Another observation is that six misclassified humans have a median time difference between consecutive activity types ($\mathsf{DCAT}_{\mathsf{median}}$) of 0.001h, which corresponds more to bot behaviour than to human behaviour (see in Table 6.8).

6.5. Threats to validity

We follow the structure recommended by Wohlin et al. (2012) to discuss the main threats to validity of our research, and their potential consequences.

Construct validity examines the relationship between the theory behind the experiments performed and the observations found. This threat is mainly related to correctness of the dataset used in the experiments. A possible such threat is that contributors in the ground-truth are not labelled correctly. This situation is very unlikely to happen since we followed a multi-rater labelling process that resulted in an almost perfect inter-rater agreement (Cohen's kappa $\kappa = 0.91$ (Jacob, 1960)). However, we cannot exclude that the ground-truth dataset contains so-called *mixed accounts* (i.e., accounts having a combination of human and bot activities) (Cassee et al., 2021).

Another threat to construct validity is that the ground-truth dataset is biased, by construction, towards bots having "bot" in their name: 67.1% of all bots in the dataset (i.e., 694 out of 1,035) contain this substring in their contributor name, which is likely considerably more than what one could expect in practice. While this higher proportion of bots having "bot" in their name cannot affect the performance of BIMBAS since it does not rely on this feature to detect bots, it may have led to an overestimation of the performance of bot identification approaches such as NBH and BotHunter that make use of that feature.

Internal validity concerns choices and parameters of the experimental setup that could affect the results of the observations. We strived to follow machine learning best practices during model construction, training, testing and evaluation. We relied on a state-of-the-art machine learning library (namely sklearn) to conduct the experiments. As such, our experimental setup is unlikely to have biased the results we obtained.

Another internal threat to validity is the criterion we used to select the repositories for executing BoDeGHa and BoDeGiC. We selected, for each contributor, the repository in which the contributor was the most active (in terms of events generated). A different selection criterion could lead to different performance results and to a different number of "unknown" predictions. However, none of these approaches explicitly define guidelines to decide on a repository whenever a contributor is active in more than one repository.

External validity concerns the degree to which the conclusions we derived are generalisable outside the scope of this study. The main threat to external validity is that BIMBAS was created and evaluated with activity sequences obtained from public GitHub repositories (because GitHub API only returns public events). Since activities in private repositories may differ in their type, frequency and order we cannot make any claims on the performance of BIMBAS on contributors in private repositories. Similarly, BIMBAS cannot be applied "as is" to other collaborative development platforms (e.g., GitLab, Gitea or BitBucket). Even though such platforms are mostly based on the same principles, and even if the technical process we followed to create BIMBAS is likely to be applicable to these platforms, there could be differences in the APIs, the activity pace, activity types of contributors, and the ways bots interact with repositories and contributors. As a consequence, it is very likely that a new classification model would need to be trained to take these differences into account.

Conclusion validity concerns whether the conclusions derived from the analysis are reasonable. Since our conclusions are mostly based on quantitative observations and are supported by the usual performance metrics to evaluate machine learning classifiers, they are unlikely to be affected by such threats.

6.6. Summary

This chapter completed **Goal 2** of this thesis by developing a new bot identification model that leverages a wide range of activity types performed by contributors in GitHub. First, we created a new manually labelled ground-truth dataset of bots and human contributors in GitHub. Second, with the help of a test set (40% of contributors) that was not used to train the new bot identification approach, we identified the limitations of existing bot identification approaches in terms of their performance and efficiency. Third, we proposed BIMBAS, a novel bot identification approach based on activity sequences. To create BIMBAS, we followed a grid-search 10-fold cross-validation on the training set (60% of contributors) and compared the results obtained by 13K+ combinations of classifiers and their hyperparameters. Through this process, we found a Gradient Boosting model and its associated hyperparameters to be the top performer. We applied the recursive feature elimination technique to remove unimportant features, retaining 38 features. We then evaluated the performance of BIMBAS on the test set.

Overall, the performance of BIMBAS is comparable to the best existing bot identification approaches, making it a good candidate for performing large scale analysis and to be implemented as part of a tool. This will be the goal of the next chapter.

CHAPTER 7

Using BIMBAS in practice

This chapter addresses **Goal 3** of this thesis. Chapter 6 showed that BIMBAS exhibits good performance in identifying GitHub bot contributors. In this chapter, we use BIMBAS in practice (i) by performing a large scale analysis, and (ii) by integrating BIMBAS as part of a command-line tool to enhance its usability.

There has been little research on how GitHub's automation mechanisms (presented in Chapter 2) are used in the context of large software ecosystems composed of a large community of collaborating contributors. Conducting such studies helps to understand the roles and dynamics of bots in large software ecosystems. Section 7.1 is based on my publication (Chidambaram & Mens, 2025) and uses BIMBAS to conduct a large scale analysis of bots in *NumFOCUS*, a large open-source software ecosystem for data science. This section reveals the differences in activity patterns between bots and humans on the one hand, and between different bot categories on the other hand. Section 7.2 is based on our publications (Chidambaram et al., 2025, 2024) and integrates BIMBAS into RABBIT an open-source command-line tool that takes in a set of contributor names and determines their type. This section details the functioning of RABBIT and shows that by an order of magnitude more efficient than the bot identification approaches covered in Section 6.3.

7.1. A large scale analysis with BIMBAS

This section partly addresses **Goal 3** of this thesis. It is based on my peer-reviewed publication in the International Workshop on Bots in Software Engineering (BotSE) 2025 (Chidambaram & Mens, 2025). This section presents a first-of-its-kind quantitative observation of how bot mechanisms (bot accounts, GitHub Apps and GitHub internal automation services) are used in a large OSS ecosystem hosted on GitHub. We selected as case study *NumFOCUS*,[1] a non-profit organisation supporting and promoting over 50 open-source software projects for data science at the moment of writing this dissertation. It includes very popular projects such as *NumPy*, *Pandas*, *Matplotlib* and so on.

To analyse its community of contributors, we rely on the public GitHub events made by contributors having participated in repositories belonging to the GitHub organisations of *NumFOCUS* projects. During three months, we observed their GitHub events to quantify the specific activity types of contributors as defined in Section 5.1. We explore and identify differences in activity patterns between the three GitHub bot mechanisms (GitHub internal automation services, GitHub Apps, and bot accounts) and the human contributors. A replication package can be found on `https://doi.org/10.5281/zenodo.14415595`.

7.1.1 Extracting activity sequences for contributors

To observe the use of bots in *NumFOCUS*, we rely on the public events in all repositories of GitHub organisations associated to *NumFOCUS* projects. From these public events, we identify all involved contributors (bots and humans). We exclude any events made by these contributors in repositories and organisations that do not belong to *NumFOCUS*, as they are considered out of scope. We consider a three-month observation period from July to September 2024.

We identified 60 GitHub organisations corresponding to *NumFOCUS* projects, and the 1,626 GitHub repositories belonging these organisations. For example, the dataset contains public events for all 12 GitHub repositories for the numpy GitHub organisation corresponding to the NumPy project.[2] We excluded the conda-forge organisation containing tens of thousands of repositories for the packages (called recipes) of the Conda package manager, since we consider it to be a separate packaging ecosystem.

As a data source, we relied on the dataset provided by Hourri et al. (2025). It contains all 358,451 raw public events performed by 21,957 contributors in the identified repositories during the observation period. 14,351 contributors (65%) performed only a single event, of which 9,788 corresponded to *Starring repository* and 2,072 corresponded to *Forking repository.* We decided to exclude such peripheral contributors that contribute very little to the ecosystem. To do so, we removed a very long tail of 20,445 contributors (93.1%) involved in less than 10 events in *NumFOCUS* repositories. This resulted in a filtered dataset of 322,317 events (89.9%) performed by 1,512

[1] `https://numfocus.org`

[2] See `https://github.com/orgs/numpy/repositories`.

contributors.

From the obtained events, we derived more meaningful higher-level activity types, using the mapping provided in Section 5.1.2. The activity dataset contains 282,921 activities belonging to 23 different activity types, performed by 1,512 contributors in 1,315 repositories of 59 GitHub organisations.

7.1.2 Bot detection

Our aim is to explore differences in activities between various bot mechanisms and human contributors participating to *NumFOCUS* organisations and repositories on GitHub. This requires identifying the type of each contributor.

Determining bot actors corresponding to GitHub Apps or internal automation services is easy. As explained in Section 3.2.2, the GitHub REST API `users` endpoint marks their type as "Bot", and their account name always ends with the '[bot]' suffix. In this way, we identified 4 internal automation services and 13 GitHub Apps.

Bot accounts are considerably more difficult to identify. The `users` endpoint marks their type as "User", making them indistinguishable from human accounts. Researchers have therefore proposed heuristics and classification models for bot identification (see Section 3.2). Based on the comparison of bot identification approaches in Section 6.4.4 (Table 6.7 on page 86) we combine the best three to identify bots: a name-based heuristic, BIMBAS and BotHunter. We started with the name-based heuristic to identify bots, by checking if their lowercased account name contains the string "bot". This resulted in 20 potential bot accounts. We manually excluded false positives (e.g., common human surnames such as "Abbot"), resulting in 19 confirmed bot accounts. We processed the remaining user accounts using BIMBAS developed in Chapter 6. As BIMBAS is based on a Gradient Boosting classifier, it provides a probability score for each contributor, indicating the likelihood of being a bot and the likelihood of being a human. We retained only those 817 accounts for which BIMBAS provided a probability score of at least 0.85 for either type. We applied BotHunter (Abdellatif et al., 2022) on them to check whether its prediction agreed with BIMBAS. In case of disagreement, we manually verified whether the account should be classified as bot. This allowed us to identify 15 more bot accounts, leading to a total of 34 bot accounts (of which 19 containing "bot" in their name) and 802 human accounts.

Table 7.1 provides the characteristics of the final dataset of all 853 considered *NumFOCUS* contributors and their associated activity sequences, grouped by contributor type: 4 internal automation services, 13 GitHub Apps, 34 bot accounts and 802 human accounts. While 94% of all considered contributors are humans (802 out of 853), they account for only 53.4% of all activities (133,173 out of 249,185). This suggests that bots are more active, which is confirmed by their median number of activities, that is considerably higher than for humans.

Table 7.1: Breakdown of contributor types in dataset.

	internal automation services	GitHub Apps	bot accounts	human accounts	**all**
#contributors	4	13	34	802	853
#activities	91,381	4,859	19,772	133,173	249,185
median #activities	1,116	115	104	56	58
#repositories	414	239	234	1,108	1,169
median #repositories	84	5	2	4	4
#organisations	55	46	45	58	59
median #organisations	24.5	4	1	1	1

7.1.3 Prevalence of bots in *NumFOCUS* repositories and organisations

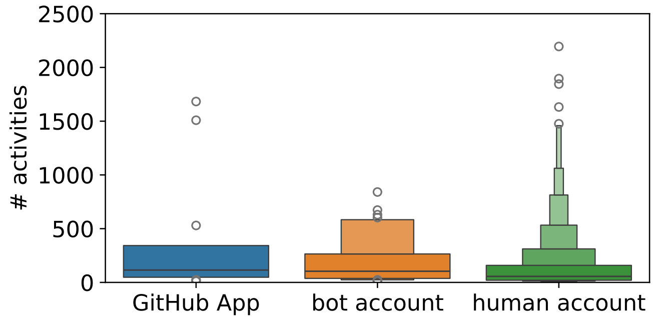
58 of the 59 considered *NumFOCUS* organisations (98.3%) automate their tasks with bots. Only the Open-MBEE organisation (an open source collaborative engineering system) does not use any bots to automate its tasks. The identified bots are involved in 583 of the 1,169 repositories (49.9%) belonging to these organisations, and they are the most active contributor in 155 of them (26.6%). The median values in Table 7.1 reveal that internal automation services and GitHub Apps participate in more organisations and repositories than human accounts. In contrast, bot accounts tend to restrict themselves to fewer repositories belonging to a single organisation.

To observe the difference in number of activities, repositories and organisations that contributor types are involved in, we plot the distribution of all contributors (per contributor type) in Fig. 7.1a, Fig. 7.1b and Fig. 7.1c respectively with *boxen plots* (a.k.a. letter-value plots (Hofmann et al., 2017)) as they provide more detailed information about the distribution than regular *box plots*. The procedure followed by boxen plot to represent the data is explained in Section 5.2.2. We excluded the four internal automation services as there are not enough to have a meaningful visualisation.

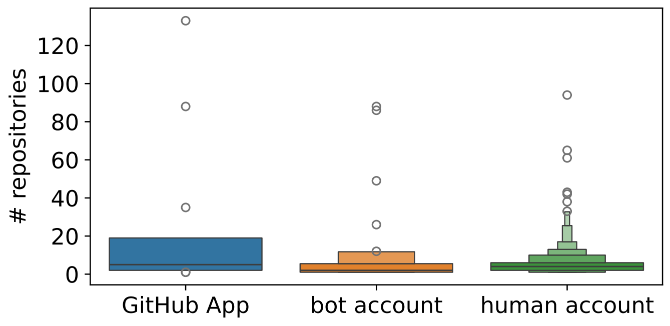
The distributions are skewed for the number of repositories contributors are involved in, with a median of five repositories for Apps, two for bot accounts, and four for human accounts. The 75^{th} percentile confirms that Apps are involved in considerably more repositories, with a value of 19 compared to only 5.5 for bot accounts and six for human accounts. This is likely because Apps are readily available and can be installed easily from the GitHub Marketplace to automate activities in any repository.

The findings for the distributions of the number of organisations are similar to those for repositories, with bot accounts and human accounts being involved in very few organisations (median of 1), and Apps being involved in considerably more organisations (median of 4 and 75^{th} percentile of 10).

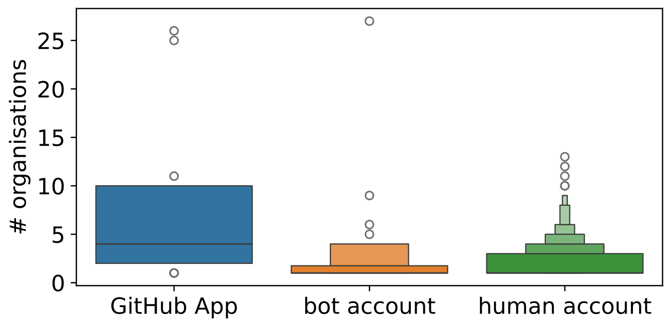
Focusing on the four internal automation services we observe that: (1) github-actions[bot], acting on behalf of the GitHub Actions workflows used by a repository, accounted for 89,117 activities in 319 repositories of 45 organisations; (2) dependabot[bot] that automates dependency updates had 1,937 activities in 163 repositories



(a)



(b)



(c)

Figure 7.1: Boxen plots of number of (a) activities (y-axis is limited to 2,500 to enhance readability), (b) repositories, and (c) organisations for *NumFOCUS* contributors grouped by contributor type.

of 47 organisations; (3) github-merge-queue[bot] that provides a merge queue for PRs had 295 activities in four repositories of a single organisation; and (4) github-advanced-security[bot] that improves and maintains code quality and security performed 32 activities in five repositories of four organisations.

Among the 13 GitHub Apps, the three most active ones were: (1) codecov[bot] that reports on code coverage, with 1,683 activities in 88 repositories of 25 organisations; (2) pre-commit-ci[bot] that provides a CI service for pre-commit framework, with 1,509 activities in 133 repositories of 26 organisations; (3) renovate[bot] that automates dependency updates, with 530 activities in five repositories of three organisations.

Among the 34 bot accounts, the three most active ones were: (1) editorialbot with 13,346 activities (of which 6,419 (48.1%) are of type *Commenting issue*) in only 4 repositories belonging to the openjournals organisation; (2) bioc-issue-bot with 841 activities (of which 768 (91.3%) are of type *Commenting issue*) in a single repository belonging to the Bioconductor organisation; (3) conda-bot with 673 activities (of which 332 (49.3%) are of type *Pushing commit*) in 26 repositories belonging to the conda organisation.

Finding: Bots are prevalent in *NumFOCUS*, contributing to 98% of its organisations and 50% of its repositories, and being the most active contributors in 27% of these repositories. Human accounts and bot accounts tend to restrict their activities to few organisations. editorialbot and bioc-issue-bot are the most active bot accounts, being involved mostly in *Commenting issue*. GitHub Apps tend to be involved in more repositories belonging to more organisations than bot accounts. The most active bots are the GitHub internal automation services github-actions[bot] and dependabot[bot].

7.1.4 Which activity types are contributors involved in?

While RQ1 focused on the number of repositories and organisations, RQ2 focuses on the number of activities and activity types per contributor type. Table 7.1 revealed that bots have a higher median number of activities than human accounts. Fig. 7.1a reveals a skewed distribution of number of activities performed by contributors, grouped by contributor type. To determine whether different contributor types tend to be involved in different types of activities in *NumFOCUS*, (i) we quantify, per contributor type, the proportion of all activities carried out per activity type as visualised in Fig. 7.2, and (ii) we analyse the difference in proportion of activities per activity type group between humans and bot contributors in Fig. 7.3 and between different internal automation services, Apps and bot accounts in Fig. 7.4. We grouped activity types into four different groups, namely PR, issue, repository and commit for ease of visualisation and understanding.

For each contributor type there are outliers that are considerably more active than all others. Two human accounts were involved in 12,321 and 5,206 activities (respectively 9.3% and 3.9% of all human activities). A single bot account, editorialbot was responsible for 13,346 activities (67.5%) of all activities performed by bot accounts. codecov[bot] and pre-commit-ci[bot] were responsible for 1,683 and 1,509 activities (34.6% and 31.1% of all activities performed by Apps). The internal automation service github-actions[bot] was responsible for 89,117 activities, accounting for 97.5% of

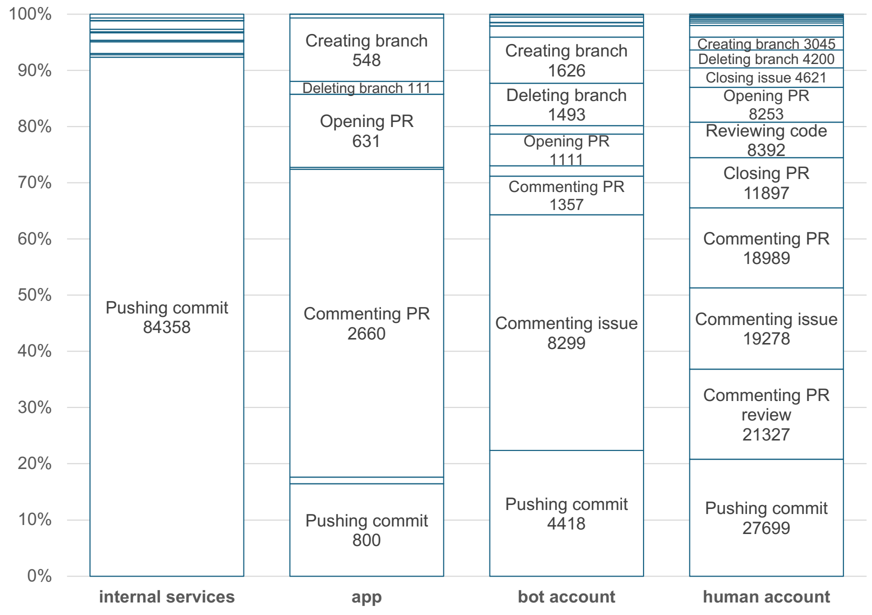


Figure 7.2: Proportion of activity types performed per contributor type. Only activity types with a high enough proportion of activities are labeled along with the number of activities of that type.

all activities performed by internal automation services.

Human accounts are involved in all 23 activity types. A single human accounted for 43.9% of all *Pushing commit*, an activity carried out by 401 humans. 1,928 human accounts performed *commenting* (on issue, PR or PR review) (44.7% of all human activities). Review-related activities (i.e., *Commenting PR changes* and *Reviewing code*) were virtually absent for bots, while they account for 22.3% of all human activities. Overall, as observed from Fig. 7.3, humans majorly perform PR-related activities (52%) compared to that of commit- (20.9%), issues- (20.2%) and repository-related activities (6.9%).

Bot accounts are involved in 16 activity types. They are mostly *Commenting issue* (42% by 10 bots) while this type only accounted for 1.2% of all activities for Apps. The secondmost frequent activity for bot accounts was *Pushing commit* (22.3% by 13 bots). Both activity types are predominantly performed by editorialbot, accounting for 77.3% of all *Commenting issue* and 61.6% of all *Pushing commit*. Other major activity types are *Creating branch* (8.2%), *Deleting branch* (7.6%), *Commenting PR* (6.9%), and *Opening PR* (5.6%). As observed from Fig. 7.4, bot accounts majorly perform issue-related activities (45.5%) compared to that of commit- (23.3%), repository- (16.9%) and PR-related activities (14.3%).

GitHub Apps performed 10 activity types. The most frequent type is *Commenting PR*, carried out by ten Apps. Five Apps even exclusively performs this activity.

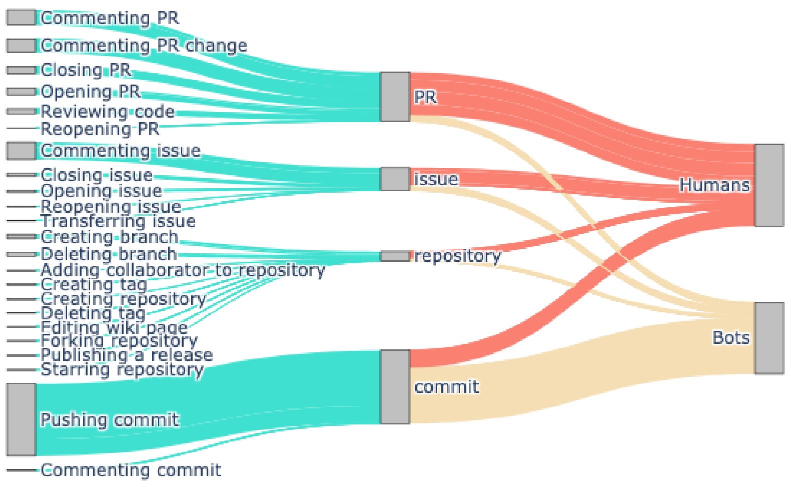


Figure 7.3: Sankey diagram of the distribution of activities performed by humans and bot contributors.

This is for example the case for codecov[bot] that accounts for 63.3% of all *Commenting PR* activity performed by Apps, mainly for posting code coverage analysis reports computed by Codecov. Four Apps are involved in a combination of *Creating branch*, *Opening PR* and *Pushing commit*. Overall, Apps are majorly involved with PR (68.1%) related activities compared to that of commit (16.5%), repository (13.6%) and issue (1.9%) related activities.

Internal automation services are involved in 15 activity types, with *Pushing commit* being the predominant type (92.3%). As observed in Fig. 7.4 99.8% of this activity type is due to github-actions[bot] that performed 84,200 *Pushing commit* activities. dependabot[bot] is more diverse, being involved in seven activity types, of which 32.6% *Creating branch* and 32.3% *Opening PR*.

Identifying the differences in the proportion of activities performed by different contributor types was possible only because of our event type to activity type mapping presented in Section 5.1. Without this mapping, the conclusions would have been different. For example, the activities *Commenting PR* and *Commenting issue* would have been grouped under the same event type IssueCommentEvent, and the differences in proportion of these activity types between bot accounts and GitHub Apps would not have been visible. Similarly, *Closing PR* and *Opening PR* would have been grouped under the event type PullRequestEvent, and we would have observed a different contribution pattern at low-level.

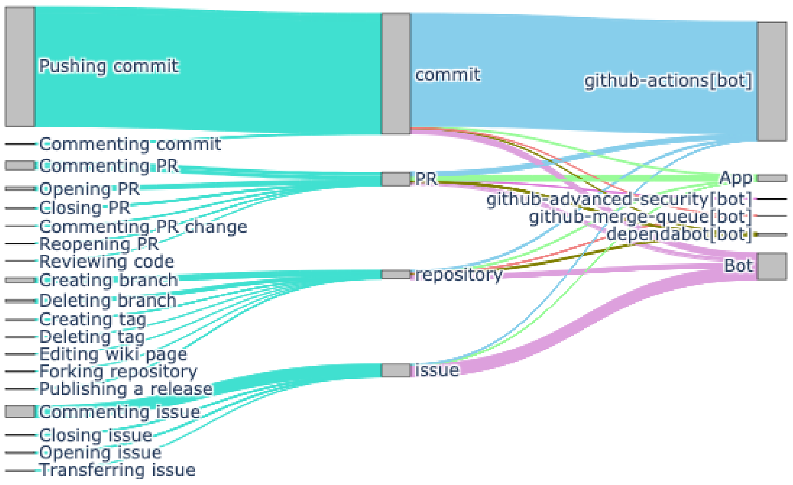


Figure 7.4: Sankey diagram of the distribution of activities performed by different internal automation services, Apps and bot accounts.

Finding: Bots are more active than humans. Humans are involved in more activity types than bots. Bot accounts are involved in more activity types than Apps. The activity distribution is heavily skewed for all contributor types. Internal automation services are majorly involved with commit-related activities, while for bot accounts it is mostly issue-related activities, and for Apps and humans mostly PR-related activities.

7.1.5 Which organisations are contributors involved in?

Similar to Section 7.1.4, we analysed the number of activities per activity type performed by different bot categories in each of the considered *NumFOCUS* organisations. This visualisation is given in Fig. 7.5.

We observe that, among the 84,814 *Pushing commit* activities performed by github-actions[bot] in *NumFOCUS* organisations, 75,659 of them (89.2%) are performed in the Bioconductor organisation. *Pushing commit* is also the most frequent activity type (98.8%) performed in Bioconductor, an organisation distributing more than 2,000 R packages related to computational biology and bioinformatics. As such, it can be regarded as a package manager of its own, similar to the conda-forge organisation.

We also observed 13,377 activities performed by bots in the openjournals organisation, which aims to collect and share open-source and open-access journals. 99.8% of its activities are performed by editorialbot, a bot account used for automating the editorial process of open access journals. A majority (52%) of the activities performed by editorialbot in openjournals organisation are related to managing issues.

These observations complement the observation in Section 7.1.3 that Bioconductor and openjournals are the two organisations with the highest number of bot activities.

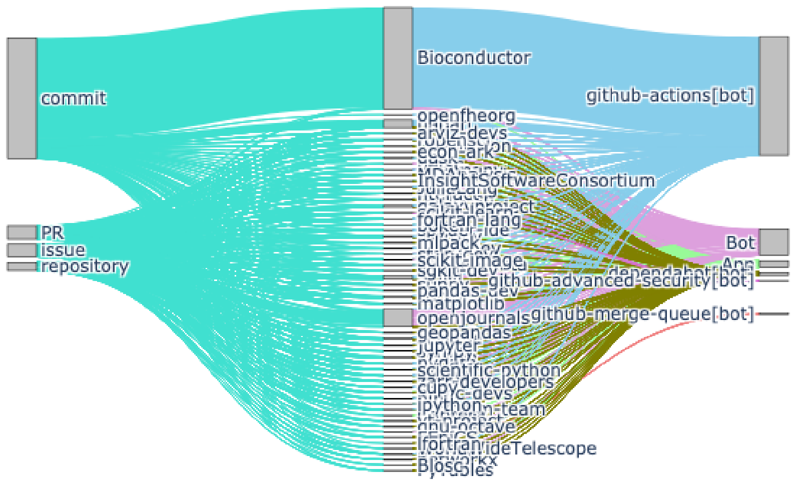


Figure 7.5: Sankey diagram of the number of activities performed by different bot categories in *NumFOCUS* organisations

In the remainder of this subsection, we continue the analysis while excluding these "outlier" organisations.

Fig. 7.6 is the modified version of Fig. 7.5 after excluding Bioconductor and openjournals. We observe that bot activities are now more equally distributed over the different organisations: each of them involves less than 8.8% of the bot activities, except for napari that accounts for 21.9% of all bot activities.

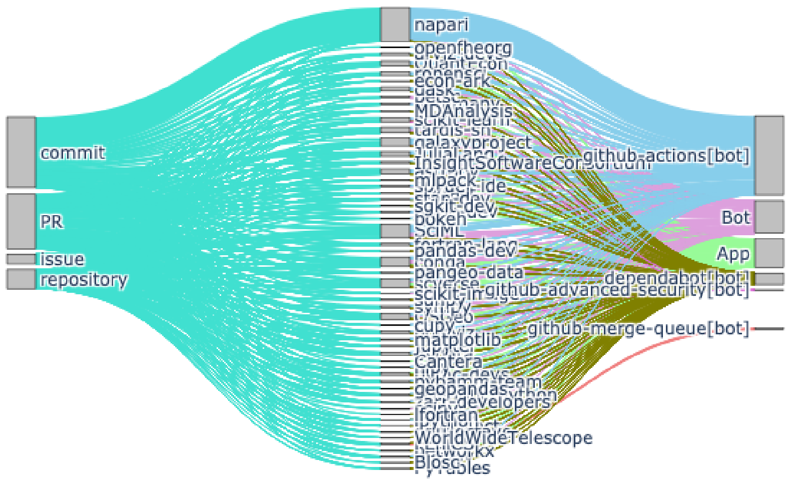


Figure 7.6: Sankey diagram of the number of activities performed by different bot categories in *NumFOCUS* organisations excluding Bioconductor and openjournals.

We also redid the analysis of Fig. 7.4 after excluding the Bioconductor and openjournals organisations. Fig. 7.7 now paints a quite different picture. Among 26,091 activities performed by bots, only 13,440 (51.5%) activities are performed by github-actions[bot]. Similarly, among the 11,979 *Pushing commit* activities performed by bots, 9,137 activities (76.3%) are performed by github-actions[bot]. Also, each bot category performs a similar number of PR-related activities in these organisations. 3,781 (14.5%) PR-related activities are performed by internal automation services, 3,291 (12.6%) by GitHub Apps and 2,119 (8.1%) by bot accounts.

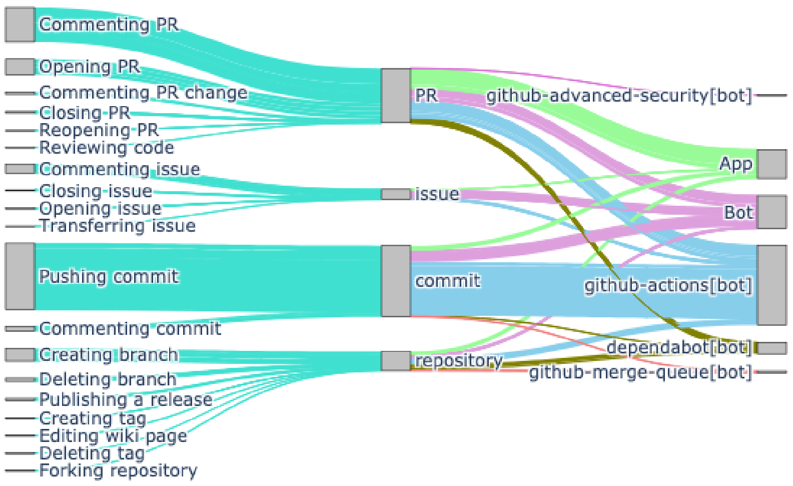


Figure 7.7: Sankey diagram of distribution of activities per activity type performed by different bot categories in *NumFOCUS* organisations excluding Bioconductor and openjournals.

Finding: Bioconductor and openjournals are the most active organisations in terms of bot contributions. After removing these organisations, bot activities tend to be more or less equally distributed across organisations. We also observed that all bot categories contribute more or less equally to PR-related activity types.

7.2. An efficient bot identification tool

In order to allow researchers and practitioners to use BIMBAS, the activity-based bot identification model proposed and evaluated in Chapter 6, we propose RABBIT, an efficient executable command-line tool to detect bots on GitHub. This section completes **Goal 3** of this thesis. It is based on the peer-reviewed publications in the (i) Journal of Systems and Software (JSS) 2025 (Chidambaram et al., 2025), and (ii) International Conference on Mining Software Repositories (MSR) 2024 (Chidambaram et al., 2024). Section 7.2.1 introduces RABBIT and explains its functioning. Finally, Section 7.2.2 evaluates and compares its efficiency against the bot identification approaches of Section 6.3, showing that RABBIT addresses their main efficiency-related limitations.

The BIMBAS bot identification model developed in Chapter 6 addresses the efficiency limitations of the other bot identification approaches presented above. This enables us to proceed integrating BIMBAS into RABBIT in Section 7.2.1 without requiring any modifications to the model.

7.2.1 Implementation of RABBIT

RABBIT is a recursive acronym for "RABBIT is an Activity-Based Bot Identification Tool". It provides a command-line interface to use the BIMBAS model trained on the full dataset. It is released as an open source project on GitHub[3] under the Apache 2.0 License. Release 2.2.0 of RABBIT was used for the experiments in the current section. We published an earlier version of RABBIT, implementing an XGBoost classification model based on a more limited set of features and trained on a significantly smaller dataset, in the Data and Tool Showcase Track of the International Conference on Mining Software Repositories (Chidambaram et al., 2024).

RABBIT can be installed using Python's package manager `pip` with `pip install git+https://github.com/natarajan-chidambaram/rabbit`.

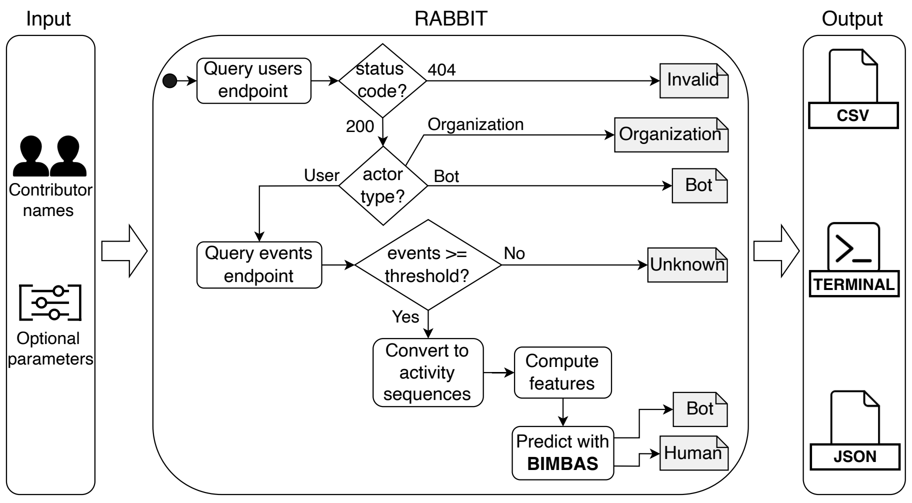


Figure 7.8: Schematic representation of RABBIT.

The functioning of RABBIT is schematically represented in Fig. 7.8. The mandatory input is a list of contributor names. These names can be provided directly on

[3]`https://github.com/natarajan-chidambaram/rabbit`

the command line or through a text file. RABBIT also provides several optional parameters to change its default behaviour (number of queries, number of events, etc.) The output can be displayed on the standard output (by default) or stored in a CSV or JSON file.

RABBIT is a user-friendly and modular tool that has dedicated modules for each step in the bot identification process: (1) a Python script for extracting the required fields from events provided by the `events` endpoint; (2) a Python script for identifying activities from events; (3) a Python script for computing the features from activities; and (4) a joblib file containing parameters of the trained BIMBAS model. For example, if a user has the raw events (in JSON format) performed by contributors in a specific set of GitHub repositories, they can write a Python script that can take the events as input, use the second module to identify the activities, the third module to compute the features, and the fourth module to predict the type of contributor.

RABBIT will assign one of five possible types to each provided contributor name: *Human*, *Bot*, *Organization*, *Unknown* or *Invalid*. To do so, the tool follows the process depicted in the middle part of Fig. 7.8. The process starts by querying GitHub REST API's `users` endpoint to extract the value stored in the "type" field. The contributor type is predicted as *Invalid* if the contributor name does not exist on GitHub. If the contributor does exist, but the value in the "type" field is not "User" (e.g., it will be "Bot" for GitHub Apps, and "Organization" for organisations), this value is provided as output without any further processing. Only if the "type" field is "User", RABBIT queries the `events` endpoint[4] to extract up to 300 events (using at most 3 API queries) performed by the contributor during the last X days (where X is the retention period imposed by the API which has been reduced to 30 days as of 30th January 2025). The contributor type will be *Unknown* if the number of events obtained does not reach the minimum threshold set by RABBIT (which is 5 events by default). If enough events can be retrieved, the extracted event sequence is converted into an activity sequence (see Section 6.4.1), the features are computed for this activity sequence (see Section 6.4.2), and the BIMBAS classification model is used to predict the contributor type as either *Bot* or *Human*.

Listing 7.1: Example of RABBIT usage and output.

```
% rabbit --input-file names.txt
         contributor            type            confidence
 github-actions[bot]            Bot             1.0
      johnpbloch-bot            Bot             0.932
     openssl-machine            Bot             0.714
           ritchie46            Human           0.926
    juliaregistrator            Bot             0.875
          gvanrossum.           Human           0.960
              google            Organization    1.0
            renovate            Unknown         -
               gh-ci            Invalid         -
```

Along with the contributor type, RABBIT also reports on the *confidence* of its decision. The confidence score is based on the *probability* associated to each prediction made by BIMBAS. Given that BIMBAS provides the probability for a contributor to be a

[4]https://api.github.com/users/CONTRIBUTORNAME/events

"bot", the confidence score is computed as $|probability - 0.5| * 2$. Listing 7.1 shows an example of the execution of RABBIT with a list of 8 contributor names provided through the input file `names.txt`.

The mandatory input to RABBIT is a GitHub contributor's login name and/or a text file with a set of login names. When both are provided, the type for contributor names that are parsed directly as input parameters are determined before proceeding with the contributor names provided in the text file. RABBIT has several optional arguments to change its default behaviour and make it flexible. A full description of these optional arguments can be obtained by typing `rabbit --help` in the terminal. In particular,

`--key` can be used to provide a GitHub API key. This is mainly useful in order to make more than 60 queries to the GitHub API. If not provided, the tool will wait for the API rate-limit to reset, and resumes on its own after that.

`--min-events` specifies the minimum number of events that need to be considered for making the prediction. The default value is 5, and the maximum value is 300, as it corresponds to the maximum number of events provided by the `events` endpoint at the time of writing this dissertation.

`--min-confidence` provides the minimum confidence on the contributor type prediction to stop further querying (default is 1). Querying will be stopped irrespective of the confidence threshold, if the threshold number of queries provided through `--max-queries` is reached.

`--max-queries` specifies the number of API queries that should be made for each account, either 1, 2 or 3 (default is 3). If the threshold confidence value provided in `--min-confidence` is reached, further querying will be stopped.

`--verbose` outputs additional information, i.e., the number of events made by the contributor in GitHub and values of all features that were used to make the prediction.

`--csv` and `--json` can be used to save the output as a comma-separated-values or a JSON file, respectively. If not provided, the results will be directly printed on the terminal.

`--incremental` can be used if the user wants to see the results as soon as RABBIT determined the type for each contributor. This avoids the need to wait for RABBIT to determine the type of all the provided contributors before proceeding further.

7.2.2 Comparing RABBIT's efficiency with existing approaches

RABBIT benefits from the good model performance of BIMBAS, while at the same time addressing all limitations reported in Section 6.3 for the existing bot identification approaches. For instance, RABBIT supports a wider range of activity types and is able to determine the type of many more contributors than BoDeGHa or BoDeGiC.

To evaluate the efficiency of RABBIT, we applied it on the same test set of 860 contributor names as those that were used for evaluating the other bot detection approaches in Section 6.3. To determine the type of these 860 contributors, RABBIT required 22 minutes and 112 MB of downloaded data. Only 2,426 API queries were required to make all predictions, staying well below GitHub's API rate limit of 5,000 queries per hour and per API key. By extrapolation, this means that RABBIT can process 1,772 contributors on average before reaching the rate limit.

Table 7.2: Efficiency comparison of RABBIT against existing approaches on the test set of 860 unseen contributors.

approaches	data downloaded	time	API queries
NBH	-	0.01 sec	-
BoDeGHa	3.83 GB	7.7 h	10,222
BoDeGiC	23.3 GB	22.1 h	-
BotHunter	0.261 GB	20.8 h	37,240
RABBIT	0.112 GB	22 m	2,426

Table 7.2 compares RABBIT's efficiency to the existing bot identification approaches, highlighting their limitations in terms of execution time, data downloaded, and number of required API queries. In terms of execution time, RABBIT is more than an order of magnitude faster than BoDeGHa (21×), BotHunter (57×) and BoDeGiC (60×). RABBIT requires considerably less data to be downloaded compared to BoDeGHa (34×) and BoDeGiC (208×). RABBIT uses an order of magnitude less API queries than other approaches relying on the GitHub API, namely BoDeGHa (4×) and BotHunter (15×).

In summary, RABBIT can be used for considerably larger sets of contributors, since it runs faster, uses less API queries and requires less data, while still achieving a performance that is comparable to the best existing bot identification approaches.

7.2.3 Limitations

A first limitation of RABBIT is that it relies only on an account's public events, since the API does not provide access to events in private repositories. The same limitation is applicable to the existing bot identification approaches that depend on the events performed by a contributor in GitHub. Second, RABBIT only relies on information obtained from the GitHub REST API's `events` endpoint. One could consider extracting other activity types through additional endpoints, such as `issues` endpoint, that provide issue-specific information such as locking, unlocking and labelling issues. But this would require more API queries, more data to be downloaded, and more execution time. Third, since the `events` endpoint returns the most recent events only, RABBIT is unable to predict accounts that were not recently active. Finally, as for any known bot identification tool or even for human raters, it is impossible to predict so-called *mixed accounts* that combine both human and bot behaviour (Cassee et al., 2021).

7.3. Summary and conclusions

This chapter addressed **Goal 3** of this thesis by leveraging the practical use of the BIMBAS bot identification model. While many bot identification approaches have seen the light in recent years, there is little large-scale empirical evidence on how bots act “in the wild” in large ecosystems of interrelated software repositories on GitHub. We therefore carried out a case study on the *NumFOCUS* ecosystem for data science.

We performed an empirical analysis of the presence and use of internal automation services, bot accounts and GitHub Apps in *NumFOCUS*, a large open source software ecosystem for data science. Analysing differences in the type and frequency of activities being performed by these contributors, we observed that bots behave differently than humans, and that the three types of bots exhibit different activity patterns. Internal automation services were also observed to be active in considerably more repositories and organisations than other automation mechanisms. These promising preliminary insights call for the need for more in-depth studies on the use and complementarity of the different automation practices provided by GitHub. This may help to uncover the roles and values of such practices during collaborative development, and how their use evolves over time and across organisations and projects. There is also a need to carry out case studies on other large ecosystems, since the observed findings may not generalise beyond *NumFOCUS*.

As another way to leverage the practical use of BIMBAS, we implemented RABBIT, a command-line tool to allow researchers and practitioners to identify bots in GitHub. RABBIT is more efficient than previous bot identification approaches. While still achieving a comparable performance, it can be used for considerably larger sets of contributors since it runs an order of magnitude faster, uses an order of magnitude less API queries and requires less data to be downloaded.

Based on one’s specific needs, other bot identification approaches could be favored. If accuracy is not a crucial factor, then NBH should be favored since it is really easy to implement and the fastest bot identification approach so far. If higher model performance would be preferred over a faster execution time, for example in the context of some empirical research study, BotHunter might be favored since it had less misclassified cases on our test set than RABBIT. However, as explained in Section 6.4.4, this is likely to be a consequence of the high proportion of bots that have “bot” in their name in the ground-truth dataset, and the strong reliance of BotHunter on the NBH heuristic.

Overall, RABBIT offers a simple and easy way to apply BIMBAS in practice, by processing thousands of contributors per hour while staying under GitHub’s API hourly rate limit, making it suitable for large scale analysis.

CHAPTER 8

Conclusion

Software developers often join together to develop complex software applications in a collaborative manner. Such collaborative software development practices have given rise to development and emergence of social coding platforms. GitHub, the biggest social coding platform allows developers and maintainers to frequently use automation mechanisms to automate repetitive, error-prone and effort-intensive tasks.

Chapter 2 presented various GitHub automation mechanisms such as automated workflows, internal automation services, bot accounts and GitHub Apps, with the latter three collectively called bots. As discussed in Chapter 1 and Chapter 4, bots are widely used in GitHub and belong to the most active contributors in GitHub. As presented in Chapter 3, determining whether a contributor corresponds to a bot or a human is important in various empirical studies. Many bot identification approaches have been proposed. However, evaluating the existing approaches in terms of their performance (Chapter 6) and efficiency (Chapter 7) revealed that these approaches suffer from various limitations. This thesis provided details on the empirical analyses that were performed to identify the prevalence of bots in GitHub repositories and distinguishing behavioural features between bots and humans, datasets of contributor activity sequences and ground-truth on contributor type that were created, and the classification models and a tool that were developed or improved for better bot identification in collaborative software development on GitHub. More specifically, this thesis provided the necessary support for the following thesis statement:

Thesis statement:
It is important to identify bots in GitHub repositories. Existing bot identification approaches suffer from various limitations. We propose improved bot identification approaches that overcome these limitations.

The remainder of this chapter is structured as follows. Section 8.1 summarises the contributions of this dissertation. Section 8.2 discusses the limitations. Finally, Section 8.3 sheds light on the future research perspectives that this dissertation has opened up.

8.1. Contributions

This section summarises the contributions that were made through this dissertation. It describes the datasets that were created, the empirical studies that were performed, and classification models and the tool that were developed and improved.

8.1.1 Datasets created

As the events provided by GitHub's REST API `events` endpoint do not explicitly correspond to the actual activity types performed by a contributor, in Chapter 5 we created a mapping to identify higher-level activities from raw public events. Based on this mapping, we created a dataset of more than 833 thousand activities performed by 385 bots and 616 humans in GitHub (Chidambaram et al., 2023a). Inspired by the event to activity mapping, a researcher from our lab developed a more fine-grained mapping that represents even more closely the actual activities of GitHub contributors (Hourri et al., 2025).

In order to train and evaluate a machine learning-based bot identification model, we also created a ground-truth dataset of bots and human contributors in GitHub (Chidambaram et al., 2025). To do so, we relied on the curated dataset that we presented in Chapter 5, and the contributors that were used in studies by Wyrich et al. (2021), Cardoen et al. (2024) and Abdellatif et al. (2022). The resulting ground-truth consisted of 2,150 contributors of which 1,035 are bots and 1,115 are humans.

8.1.2 Bot identification models

This section highlights the contributions made towards bot identification approaches in this thesis. Fig. 8.1 summarises these contributions. The leftmost boxes present the existing bot identification approaches discussed in Chapter 3. The orange coloured boxes correspond to classification models that are developed in this thesis and violet coloured boxes represent the tools that are newly developed or improved through a PR.

In Chapter 4 we compared the performance of five existing bot identification approaches. For this, we first conducted an exploratory study on the accuracy of these

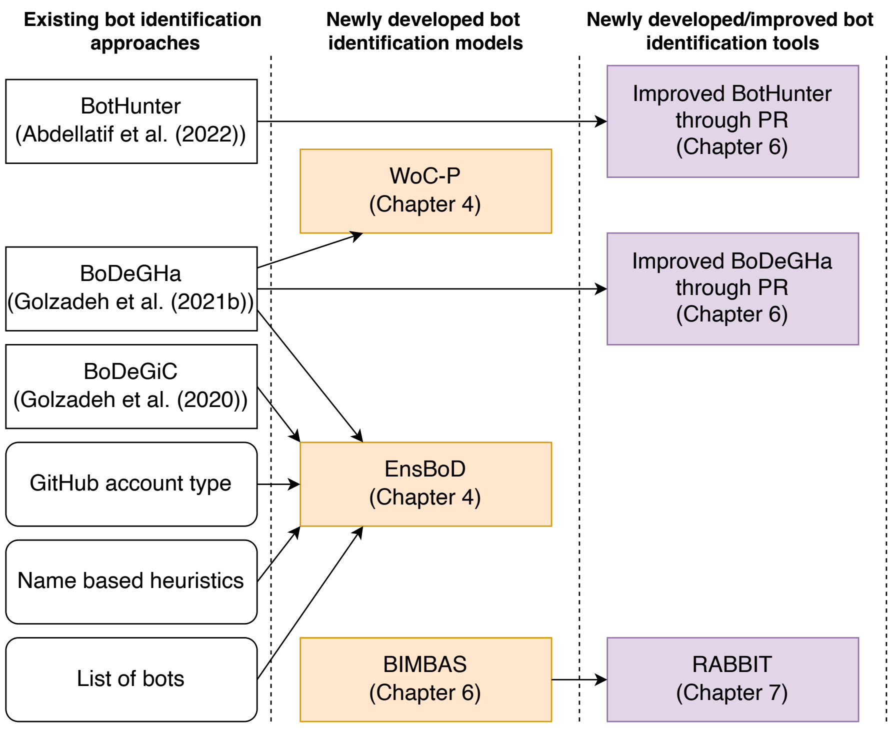

Existing bot identification approaches
Newly developed bot identification models
Newly developed/improved bot identification tools
BotHunter (Abdellatif et al. (2022))
Improved BotHunter through PR (Chapter 6)
WoC-P (Chapter 4)
BoDeGHa (Golzadeh et al. (2021b))
Improved BoDeGHa through PR (Chapter 6)
BoDeGiC (Golzadeh et al. (2020))
GitHub account type
EnsBoD (Chapter 4)
Name based heuristics
List of bots
BIMBAS (Chapter 6)
RABBIT (Chapter 7)

Figure 8.1: Contributions to bot identification approaches. Orange coloured boxes correspond to classification models and violet coloured boxes are tools that are newly developed or improved through a PR.

approaches on a dataset containing the top 20 most active contributors (in terms of number of commits) of a set of 27 large GitHub repositories. This preliminary analysis revealed that none of the compared techniques is perfectly effective in detecting all the bots in the considered set. We hypothesised that combining these five approaches could lead to an improved bot identification approach. To confirm this hypothesis, we developed an ensemble classifier, called EnsBoD, that incorporates the capability of all individual approaches. We evaluated EnsBoD through a 10-fold cross-validation process, and found that it leads to more accurate predictions (Golzadeh et al., 2022a).

Another contribution of this dissertation is a classification model that takes into account the activity of contributors in multiple repositories. As BoDeGHa works at the level of individual GitHub repositories, it led to different predictions for the same contributor in different repositories and incomplete predictions in the repositories in which the contributor was not active enough. To improve the predictions of BoDeGHa, we developed a model, called WoC-P, that relies on the wisdom of the crowd principle. More specifically, if a contributor involved in multiple repositories has different predictions, i.e., *bot* in certain repositories and *human* in others, we assume that the most frequent prediction (either *bot* or *human*) is correct, while the less frequent one is not. WoC-P improved the bot identification in the considered set of repositories, suggesting that combining predictions from multiple repositories helps to improve the performance of BoDeGHa. Additionally, we demonstrated that incomplete predictions, where a model fails to predict an account type, can be addressed by combining predictions from multiple repositories (Chidambaram et al., 2022).

The final bot identification model we developed was presented in Chapter 6 (Chidambaram et al., 2025). We compared the performance of four existing bot identification approaches on the ground-truth dataset of contributors that were created in Chapter 6. Through this comparison, we found the limitations of these approaches and highlighted the need for a more practical bot identification approach that:

- Exhibits a good model performance, so that the predictions can be reliable.
- Does not restrict itself to specific activity types.
- Can be applied at scale to classify thousands of contributors in a limited amount of time, in order to use it for large-scale studies.
- Is future-proof, by remaining applicable even when textual comments provided by bots (in issues, PRS and commits) become indistinguishable from human comments, due to the rapid advancement of LLM-based approaches.

Through various intuitions, we created a set of 45 features that correspond to behavioural features of contributors in GitHub repositories. The set contained five counting metrics related to contributor activity and eight metrics (e.g., number of activities per activity type, and duration between consecutive activities in a repository) that were aggregated based on the mean, standard deviation, median, interquartile range and Gini coefficient. Using these features we proposed BIMBAS, a novel binary classification model for GitHub bot identification based on activity sequences. To create this model, we followed a grid-search 10-fold cross-validation on the training set and compared the results obtained by more than 13,000 combinations of classifiers and their hyperparameters. Through this process, we found Gradient Boosting and its associated hyperparameters to be the top performing classifier. We applied the Recurrent Feature Elimination technique to remove features that do not contribute to the model performance, retaining 38 features. Evaluating the performance of BIMBAS on the test set revealed a performance comparable to the best existing bot identification approaches. This made BIMBAS a good candidate to be implemented as part of a tool.

8.1.3 Tool development

This thesis also contributed to the research community and open-source community by improving upon and creating new open-source GitHub bot identification tools. Fig. 8.1 summarises these contributions.

While evaluating the performance of BoDeGHa in Chapter 6, we identified some overly restrictive condition in its source code to avoid exceeding the GitHub API rate limit. We optimised the code by relaxing this condition and created a PR that has been merged into BoDeGHa's GitHub repository by its creator and is now part of the latest version of BoDeGHa.[1]

Similarly, while evaluating the performance of BotHunter in Chapter 6, we discovered that it retrieves only 30 events per API call while the GitHub API allows to retrieve up to 100 events at once. We therefore optimised the source code of BotHunter to retrieve up to 100 events per API call, thus reducing the number of API queries and reducing its execution time since the hourly API rate limit is reached less frequently. We created a PR of these code modifications to BotHunter's GitHub repository[2] which was accepted by the repository maintainer and has been integrated in the latest version of BotHunter.

To allow researchers and practitioners to use the new BIMBAS bot identification model that was developed in Chapter 6, we created an open-source command-line tool called RABBIT in Chapter 7. RABBIT takes as input a list of GitHub contributor names and provides the type of contributor (e.g., bot or human) along with the prediction confidence. Evaluating RABBIT's performance on a test set (that was not used for training BIMBAS) revelaed that RABBIT is more than an order of magnitude faster, requires considerably less data to be downloaded, and uses an order of magnitude less API queries compared to the state-of-the-art. This highlights that RABBIT can be used to process thousands of contributors per hour without exceeding the GitHub API rate limit.

8.1.4 Empirical analysis

This dissertation also presented the results that were obtained through empirical analysis. The preliminary analysis in Chapter 1 revealed that bots play an undeniable role in large collaborative software development projects. In some repositories, they belong to the most active contributors in terms of commits. We also observed that those active bots are not always identified by GitHub as GitHub Apps or internal automation services since many of them are bot accounts labelled as "User" by GitHub.

Another contribution is a large-scale empirical study on the *NumFOCUS* ecosystem (on the activities performed by 853 contributors in 1,116 repositories belonging to 59 organisations) through which in Chapter 7 we provided evidence that bots behave differently than humans, and that the three types of bots exhibit different activity patterns. We analysed the differences in the type and frequency of activities being performed by these contributors to identify how bots behave in a large ecosystem of interrelated software repositories on GitHub.

[1]`https://github.com/mehdigolzadeh/BoDeGHa/pull/25`
[2]`https://github.com/ahmad-abdellatif/BotHunter/pull/5`

8.2. Limitations and discussions

8.2.1 GitHub API

This section discusses the limitations imposed by the GitHub REST API.

A first limitation of the GitHub REST API's `events` endpoint relates to the lack of reliability of the event payload details. For example, a PushEvent reports on the number of commits pushed through the `size` and `distinct_size` fields. Unfortunately, the values indicated in these fields do not always correspond to the actual number of commits that were pushed, likely because of rebasing and commit squashing. Another example is the merge status reported in a PullRequestEvent that sometimes indicates that a pull request is merged when it is not (and vice versa). These perils are well-known by the community involved in mining software repositories when mining GitHub data (Kalliamvakou et al., 2014), and there is little we can do to address them.

A second limitation is that, the `events` endpoint reports only the public events performed by the contributors and not their activities in private repositories. So, we can not capture the complete sequences of activities performed by GitHub contributors. For the same reason, we do now know the performance of the bot identification approaches developed in this thesis on activities performed by contributors in private repositories.

A third limitation is that, GitHub's APIs are evolving over time. For example, the event types reported by `events` endpoint includes a new event called DiscussionEvent since 30th January 2025, and the event retention period has been reduced from 90 to 30 days. The studies and the results presented in this thesis is based on the API restrictions that were in-place at the time when the studies were conducted, specifically, it is based on API version 28-11-2022. Any change introduced by GitHub in the future might make BIMBAS and RABBIT be unusable or non-reproducible in the future. If the changes introduced by GitHub are not backward compatible, RABBIT will not be executable. The BIMBAS model is robust to API changes as it is a trained classification model that requires only the features based on contributor activities.

A fourth limitation is that, one can only obtain some details through the GitHub GraphQL API. For example, contributor activity on Discussions can be obtained only through the GraphQL API and not through the REST API. If one wants to develop a new approach considering a wider range of contributor activities, one could consider to use the GraphQL API. But depending on this API requires more API calls than the REST API as the GraphQL API does not have the functionality to access all the event types performed by a contributor in a single query.

8.2.2 Methodological limitations

This section details on the limitations that are based on the choices that were made while conducting studies in this thesis.

A first limitation is that we did not consider so-called mixed accounts (Cassee et al., 2021). Mixed accounts are GitHub user accounts that are shared between a human contributor and a bot contributor. In other words, mixed accounts are GitHub accounts belonging to humans allowing automated tools to use their account for performing certain tasks. Due to this mixed behaviour, we could not classify them exclusively as a bot or a human contributor. Since our goal was to identify bot contributors in GitHub, we removed such accounts from our ground-truth datasets during the inter-rater agreement process.

A second limitation is that, in Chapter 4, for each contributor, we assigned equal weights to all the predictions provided by BoDeGHa in different repositories. Alternatively, we could have provided a weight for each prediction, for example, based on their number of activities in that particular repository. This might have changed the performance of WoC-P. Also, we relied on wisdom of the crowd principle, in other words a majority vote, to group the predictions and determine the type of contributor. Depending on smarter, rule based, or advanced grouping approaches could have led to different performance scores.

A third limitation is the incomplete mapping of event types to activity types. Chapter 5 provided this mapping by combining a maximum of two event types to represent an activity type. However, some activity types can correspond to more than two event types. For example, if a user merges a PR that is created from a branch in the repository which is associated with an issue, then four events are generated: PullRequestEvent (corresponding to closing the PR), PushEvent (merging the PR), IssuesEvent (closing the associated issue), and DeleteEvent (deleting the branch from which the PR is created). All these event types correspond to the single activity *closing PR*.

Additionally, in Chapter 5, we relied on a two-second upper bound to consider if two events belong to the same activity. The upper bound of two seconds is adopted based on an experimental analysis of various event type combinations in the event logs of GitHub contributors. For example, closing an issue with a comment corresponds to a *closing issue* activity which is a combination of two events IssuesEvent and IssueCommentEvent that occur within a time window of two seconds. Similarly, merging a PR corresponds to a *closing PR* activity which is a combination of two events PullRequestEvent and PushEvent. The timestamp of both events reported by the `events` endpoint need to be within a time window of two seconds for us to consider them as a single activity. If not, they will be mapped to two different activity types. Adopting a different threshold might change the number of activities performed by a contributor. For example, reducing the threshold to less than two seconds would have led to more activities, as the number of event pairs might have reduced, and vice-versa if the threshold is increased to more than two seconds.

Identifying activity types from event types without depending on time could lead to a different number of activities. For example, one can just depend on PR number, issue number, branch name, tag name and so on to identify activities rather than keeping a threshold time to group events to a single activity type. Hourri et al. (2025) addressed the second methodological limitation and discussions mentioned above. They built further upon the mapping provided in Chapter 5 and created a more fine-grained and higher-level mapping that captures activities performed by contributors more accurately. Using a high threshold time, they combined multiple events to a single activity.

A fourth limitation is that we relied only on the `events` endpoint and did not consider activity types that can only be obtained through additional endpoints. Most of the contributor activities are captured by the `events` endpoint, but due to this limitation, it is likely that we missed some activities preformed by contributors in GitHub. For example, the `issues` endpoint provides issue-specific information such as locking, unlocking and labelling issues. Similarly, `pulls` endpoint provides review-specific information such as requested reviewers, review dismissals and so on. If RABBIT would have relied on additional endpoints, it would have required more API queries, more data to be downloaded, and more execution time.

8.3. Perspectives

This section proposes pathways for further exploration of ideas to continue this line of research.

8.3.1 Datasets

To construct our ground-truth dataset, we relied on a limited set of data sources, but with more time and effort thousands of contributors along with their ground-truth (*human* or *bot*) could be added to the dataset. Updating the dataset with emerging bots would be beneficial to have a more complete list of active bots in GitHub. Additionally, considering contributors having a mixed behaviour would open up to a third category of contributor type called *mixed* account in addition to the existing *bot* and *human*. Having a dataset that includes bots, humans and mixed accounts requires us to rely on multi-class classification models rather than binary classifiers for predicting the type of contributor.

In Chapter 5 and Chapter 6, we manually created a ground-truth dataset of contributors (*bot* or *human*). To reduce this manual effort, it would be interesting to use AI techniques, such as Active Learning (Ren et al., 2021) for labelling contributors. By adopting such techniques, one only needs to manually label certain contributors that the model finds difficult to label. This would help in creating a larger dataset of ground-truth contributors.

In Chapter 6, we had to obey GitHub's REST API's rate limitations as we relied on contributor events obtained from the `events` endpoint to train BIMBAS. The data that we can obtain in this way is less with the update made to the GitHub REST API's on 30th January 2025. This can be addressed by depending on Generative AI techniques Gozalo-Brizuela & Merchan (2024). One can fine-tune Generative AI models with events or activity sequences that contributors made in GitHub and can prompt the model to generate more events or activities. For example, the model can be used to generate event/activity sequences based on a scenario that is not present in the dataset for training bot identification models.

8.3.2 Bot identification approaches

Chapter 4 presented an ensemble bot identification model that performed better than the considered bot identification approaches. Similarly, it is worthwhile to evaluate if an ensemble model combining the strengths of the currently best performing bot identification models (e.g., combining BotHunter and BIMBAS) leads to an even better GitHub bot identification approach.

The BIMBAS bot identification model developed in this thesis does not favour any contributor type (bot or human). It considers equal weights for both false positives (incorrectly identifying humans as bots) and false negatives (incorrectly identifying bots as humans). Providing different weights might be helpful in reducing either of the false cases. For example, one could assign more weight to reduce false negatives (e.g., by lowering the default probability threshold of 0.5). As bots are less in number compared to humans in GitHub repositories, it will take less effort to manually check and correct false positives compared to searching for bots in a big list of contributors determined as humans. The optimal probability threshold could be obtained by performing a grid-search.

The existing bot identification approaches provide us only the prediction on the type of contributors, except for RABBIT that additionally provides the confidence in prediction as well. However, in the case of false negatives and false positives, it is difficult to identify the reasons for the prediction by bot identification approaches. This raises the need to use eXplainable AI (Lundberg & Lee, 2017) techniques to understand and analyse the motivation for the model's decisions. This analysis would help one to identify the reasons for the model's performance. For example, to enhance the data accordingly.

In this thesis, we identified behavioural features based on our intuition and statistically verified that they could be used to differentiate bots from humans. Alternatively, we can use other methods such as deep learning techniques (e.g., Recurrent Neural Networks (Werbos, 1990), (Mienye et al., 2024)) that can accept activity sequences and predict if a contributor is a *bot* or a *human*. Such AI techniques might capture deeper insights, identify hidden features based on the behaviour of contributors in GitHub and work based on those insights to predict the contributor type.

In this era of rapid improvements in AI and due to the rapid evolutionary nature of software development, there can be some changes in the way in which GitHub automation mechanisms are used in the future. As bot identification models are trained on current data using traditional machine learning methods, they might deteriorate in performance as time passes. This brings the need to have a model that can fine-tune and adapt itself to recent changes in GitHub contributor behaviour to identify bot contributors. For this, one can depend on continual learning (Wang et al., 2024), more specifically incremental learning (van de Ven et al., 2022) to learn over time while retaining previously learned information.

As presented in Chapter 1, GitHub has millions of developers and the ground-truth dataset of contributors that we considered in Chapter 6 is very limited. So, in order to make the classification model learn from a set of labelled samples and generalise well for the new samples, there is a need to depend on few-shot learning

(Snell et al., 2017). This AI technique does not require large datasets for training classifiers. Also, the model can be introduced to new classes with minimal labelled data. For example, *mixed* account (with minimum number of contributors labelled as mixed account) can be added to its training data and depending on the few-shot learning technique, it can be fine-tuned (instead of re-training the entire model) to identify contributors belonging to new class as well. As this approach does not need much data for its training, time for identifying contributor ground-truth can be saved.

8.3.3 Empirical analyses

One of the major contribution of this thesis is that we developed multiple GitHub bot identification approaches, which was only the first, yet important step to perform socio-technical and empirical studies that involves automation mechanisms in GitHub. The next important steps consist of understanding, studying and improving the usage of automation mechanisms in GitHub, and analysing how communities of open-source software developers make use of these automation mechanisms to collaborate, communicate and interact to efficiently develop and maintain high-quality software.

This dissertation just scratched the surface of empirical analysis that can be performed to study the usage and behaviour of different automation mechanisms in a large open-source software ecosystem. To better study their usage, behaviour and roles in general, there is a need to perform more and larger scale studies, i.e., on multiple software ecosystems, large collections of repositories, and so on.

To study the activities for which bot accounts are getting replaced by GitHub Apps or internal automation services, it is interesting to perform an evolution analysis of all automation mechanisms in GitHub. For example, one could analyse the number of bots, GitHub Apps, GitHub Action workflows and internal automation services that are used in one or many software ecosystems or a set of repositories for a period of time, and observe the rate at which each automation mechanism is replaced by another.

Another interesting study could be to identify the role of automation mechanisms in GitHub. For example, one can identify if a bot is used for a specific activity or for multiple activities. If it is used for multiple activities, one can study if it is used for the same category of activities such as issues-related (e.g., opening an issue and labelling an issue) or PR-related (e.g., closing a PR and commenting in a PR), or it is used for automating tasks across different activity categories (e.g., commenting issues and publishing releases). A possible extension of this study could be to analyse the differences in behaviour of bots within a specific set of software ecosystems, or a set of organisations with that of bots that are used in repositories that do not belong to a single organisation or ecosystem. Such a study could help to identify if bots used within organisations have a more predictable behaviour compared to that of bots used in a set of unrelated repositories. Another difference could be in terms of merged PRs. For this, empirical analaysis could be done to measure the difference in PR acceptance rate between (i) PR created by an automation mechanism that is used within the organisation/ecosystem to which the repository belongs to, and (ii)

PR created by an automation mechanism that does not belong to that organisation/ecosystem. Similarly, differences in human-bot interaction in GitHub repositories that belong to the same organisation/ecosystem and other unrelated repositories can be analysed. This can enable one to analyse how developers are interacting with known bots compared to that of unknown bots. This could pave the way for developers to develop bots with strict guidelines for their user-friendly collaboration and optimised usage.

We can use bot identification approaches to differentiate bots from humans and perform a useful analysis on how humans are interacting with bots and vice-versa. For example, to study what is the response time for bots when a human invokes it through PR or issue comment, what is the response time for humans when a bot mentions them in an issue or a PR, and so on. This study would highlight the quality of interaction between humans and automated mechanisms, understand the team dynamics and further help to assess the impact of bots in software repositories.

This dissertation considered activities performed by contributors only in GitHub. One can consider the contributor activities in other collaborative development platforms (e.g., GitLab, Gitea or BitBucket). Although such platforms are mostly based on the same principles of GitHub, there could be differences in activity pace, activity types of contributors, and the ways in which automation mechanisms are used to perform activities. The studies performed in this thesis can be repeated to other social coding platforms to observe the differences in usage of bots.

Bibliography

Abdellatif, A., Wessel, M., Steinmacher, I., Gerosa, M. A., & Shihab, E. (2022). BotHunter: An approach to detect software bots in GitHub. In *International Conference on Mining Software Repositories (MSR)*, (pp. 6–17). ACM.

Arora, R., Goel, S., & Mittal, R. K. (2017). Supporting collaborative software development over GitHub. *Software: Practice and Experience*, *47*(10), 1393–1416.

Atkinson, A. B. (1970). On the measurement of inequality. *Journal of Economic Theory*, *2*(3), 244–263.

Beschastnikh, I., Lungu, M. F., & Zhuang, Y. (2017). Accelerating software engineering research adoption with analysis bots. In *International Conference on Software Engineering, New Ideas and Emerging Results Track (ICSE-NIER)*. IEEE/ACM.

Biehl, M. (2017). *Webhooks – Events for RESTful APIs*. API-University.

Bock, T., Alznauer, N., Joblin, M., & Apel, S. (2023). Automatic core-developer identification on GitHub: A validation study. *Transactions on Software Engineering and Methodology (TOSEM)*, *32*(138), 1–29.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Brereton, R. G., & Lloyd, G. R. (2010). Support vector machines for classification and regression. *Analyst*, *135*(2), 230–267.

Butler, S., Gamalielsson, J., Lundell, B., Brax, C., Mattsson, A., Gustavsson, T., Feist, J., Kvarnström, B., & Erik, L. (2022). Considerations and challenges for the adoption of open source components in software-intensive businesses. *Journal of Systems and Software (JSS)*, *186*.

Cardoen, G., Mens, T., & Decan, A. (2024). A dataset of GitHub Actions workflow histories. In *International Conference on Mining Software Repositories (MSR)*. ACM.

Cassee, N., Kitsanelis, C., Constantinou, E., & Serebrenik, A. (2021). Human, bot or both? A study on the capabilities of classification models on mixed accounts. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE.

Chan, T. F., Golub, G. H., & LeVeque, R. J. (1982). Updating formulae and a pairwise algorithm for computing sample variances. In *Symp. Computational Statistics (COMPSTAT)*. Springer.

Chandrasekara, C., & Herath, P. (2021). *Hands-on GitHub Actions: Implement CI/CD with GitHub Action Workflows for Your Applications*. Apress.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, (pp. 785–794). ACM.

Chidambaram, N., Decan, A., & Golzadeh, M. (2022). Leveraging predictions from multiple repositories to improve bot detection. In *International Workshop on Bots in Software Engineering (BotSE)*. IEEE.

Chidambaram, N., Decan, A., & Mens, T. (2023a). A dataset of bot and human activities in GitHub. In *International Conference on Mining Software Repositories (MSR)*, (pp. 465–469). IEEE/ACM.

Chidambaram, N., Decan, A., & Mens, T. (2023b). Distinguishing bots from human developers based on their GitHub activity types. In *Seminar on Advanced Techniques & Tools for Software Evolution (SATToSE)*, vol. 3483, (pp. 31–39). CEUR.

Chidambaram, N., Decan, A., & Mens, T. (2025). A bot identification model and tool based on GitHub activity sequences. *Journal of Systems and Software (JSS)*, *221*.

Chidambaram, N., & Mens, T. (2025). Observing bots in the wild: A quantitative analysis of a large open source ecosystem. In *International Workshop on Bots in Software Engineering (BotSE)*. IEEE.

Chidambaram, N., Mens, T., & Decan, A. (2024). Rabbit: A tool for identifying bot accounts based on their recent GitHub event history. In *International Conference on Mining Software Repositories (MSR)*. ACM.

Cliff, N. (1993). Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, *114*(3).

Costa, J. M., Cataldo, M., & de Souza, C. R. (2011). The scale and evolution of coordination needs in large-scale distributed projects: implications for the future generation of collaborative tools. In *SIGCHI Conference on Human Factors in Computing Systems*, (pp. 3151–3160). ACM.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. In *International*

Conference on Computer Supported Cooperative Work (CSCW), (pp. 1277–1286). ACM.

Dabic, O., Aghajani, E., & Bavota, G. (2021). Sampling projects in GitHub for MSR studies. In *International Conference on Mining Software Repositories (MSR)*, (pp. 560–564). IEEE/ACM.

Decan, A., Mens, T., & Delicheh, H. O. (2023). On the outdatedness of workflows in the GitHub Actions ecosystem. *Journal of Systems and Software (JSS)*, *206*.

Decan, A., Mens, T., Mazrae, P. R., & Golzadeh, M. (2022). On the use of GitHub actions in software development repositories. In *International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 235–245). IEEE.

Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., & Mockus, A. (2020a). Detecting and characterizing bots that commit code. In *International Conference on Mining Software Repositories (MSR)*, (pp. 209–219). ACM.

Dey, T., Vasilescu, B., & Mockus, A. (2020b). An exploratory study of bot commits. In *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 61–65). ACM.

Dey, T., Vasilescu, B., & Mockus, A. (2020c). An exploratory study of bot commits. In *International workshop on Bots in Software Engineering (BotSE)*, (p. 61–65). ACM.

Dorfman, R. (1979). A formula for the Gini coefficient. *The Review of Economics and Statistics*, *61*(1), 146–149.

Erlenhov, L., Gomes de Oliveira Neto, F., Scandariato, R., & Leitner, P. (2019). Current and future bots in software development. In *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 7–11). IEEE/ACM.

Erlenhov, L., Neto, F. G. d. O., & Leitner, P. (2020). An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, (pp. 445–455). ACM.

Fairbanks, J., Tharigonda, A., & Eisty, N. U. (2023). Analyzing the effects of CI/CD on open source repositories in GitHub and GitLab. In *International Conference on Software Engineering Research, Management and Applications (SERA)*, (pp. 176–181). IEEE/ACIS.

Fang, H., Herbsleb, J., & Vasilescu, B. (2023). Matching skills, past collaboration, and limited competition: Modeling when open-source projects attract contributors. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, (p. 42–54). ACM.

Fischer, F., Höbenreich, J., & Grossklags, J. (2023). The effectiveness of security interventions on GitHub. In *SIGSAC Conference on Computer and Communications Security*, (pp. 2426–2440). ACM.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232.

Ghorbani, A., Cassee, N., Robinson, D., Alami, A., A. Ernst, N., Serebrenik, A., & Wasowski, A. (2023). Autonomy is an acquired taste: Exploring developer preferences for GitHub bots. In *International Conference on Software Engineering (ICSE)*, (pp. 1405–1417). IEEE.

Golzadeh, M., Decan, A., & Chidambaram, N. (2022a). On the accuracy of bot detection techniques. In *International Workshop on Bots in Software Engineering (BotSE)*. IEEE.

Golzadeh, M., Decan, A., Constantinou, E., & Mens, T. (2021a). Identifying bot activity in GitHub pull request and issue comments. In *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 21–25). IEEE/ACM.

Golzadeh, M., Decan, A., Legay, D., & Mens, T. (2021b). A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software (JSS)*, *175*.

Golzadeh, M., Decan, A., & Mens, T. (2020). Evaluating a bot detection model on git commit messages. In *Belgium-Netherlands Software Evolution Workshop (BENEVOL)*, vol. 2912. CEUR Workshop Proceedings.

Golzadeh, M., Mens, T., Decan, A., Constantinou, E., & Chidambaram, N. (2022b). Recognizing bot activity in collaborative software development. *IEEE Software*, *39*(5), 56–61.

Gousios, G., & Zaidman, A. (2014). A dataset for pull-based development research. In *International Conference on Mining Software Repositories (MSR)*, (p. 368–371). ACM.

Gousios, G., Zaidman, A., Storey, M.-A., & Deursen, A. v. (2015). Work practices and challenges in pull-based development: The integrator's perspective. In *International Conference on Software Engineering (ICSE)*, vol. 1, (pp. 358–368). IEEE/ACM.

Gozalo-Brizuela, R., & Merchan, E. E. G. (2024). A survey of generative ai applications. *Journal of Computer Science*, *20*(8), 801–818.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using Support Vector Machines. *Machine Learning*, *46*(1), 389–422.

Hann, I.-H., Roberts, J. A., Slaughter, S., & Fielding, R. T. (2002). Economic incentives for participating in open source software projects. In *International Conference on Interaction Sciences*.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2001). *The elements of statistical learning: data mining, inference, and prediction*, vol. 1. Springer.

Hata, H., Novielli, N., Baltes, S., Kula, R. G., & Treude, C. (2021). GitHub discussions: An exploratory study of early adoption. *Empirical Software Engineering (EMSE)*, *27*(1).

Hauff, C., & Gousios, G. (2015). Matching GitHub developer profiles to job advertisements. In *Working Conference on Mining Software Repositories (MSR)*, (pp. 362–366). IEEE/ACM.

Herbsleb, J., & Moitra, D. (2001). Global software development. *IEEE Software*, *18*(2), 16–20.

Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering (FOSE)*, (pp. 188–198). IEEE.

Hofmann, H., Wickham, H., & Kafadar, K. (2017). Letter-value plots: Boxplots for large data. *Journal of Computational and Graphical Statistics*, *26*(3), 469–477.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, *6*(2), 65–70.

Holmstrom, H., Conchuir, E. O., Agerfalk, P. J., & Fitzgerald, B. (2006). Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *International Conference on Global Software Engineering (ICGSE)*, (pp. 3–11). IEEE.

Hoover, E. M. (1936). The measurement of industrial localization. *The Review of Economics and Statistics*, *18*(4), 162–171.

Hourri, Y., Decan, A., & Mens, T. (2025). A dataset of contributor activities in the NumFocus open-source community. In *International Conference on Mining Software Repositories (MSR)*. IEEE.

Jacob, C. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37–46.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining GitHub. In *Working Conference on Mining Software Repositories (MSR)*, (pp. 92–101). ACM.

Kavaler, D., Sirovica, S., Hellendoorn, V., Aranovich, R., & Filkov, V. (2017). Perceived language complexity in GitHub issue discussions and their effect on issue resolution. In *International Conference on Automated Software Engineering (ASE)*, (pp. 72–83). IEEE/ACM.

Kazi Amit, H., Marcos, M., Yuan, T., Bram, A., & Steven, D. (2023). Understanding the time to first response in GitHub pull requests. In *International Conference on Mining Software Repositories (MSR)*, (pp. 1–11). IEEE.

Khatoonabadi, S., Abdellatif, A., Costa, D. E., & Shihab, E. (2024). Predicting the first response latency of maintainers and contributors in pull requests. *Transactions on Software Engineering (TSE)*, *50*(10), 2529–2543.

Khatoonabadi, S., Costa, D. E., Mujahid, S., & Shihab, E. (2023). Understanding the helpfulness of stale bot for pull-based development: An empirical study of 20 large open-source projects. *Transactions in Software Engineering Methodology (TOSEM)*, *33*(2).

Kolm, S.-C. (1976). Unequal inequalities. i. *Journal of Economic Theory*, *12*(3), 416–442.

Lambiase, S., Catolino, G., Palomba, F., & Ferrucci, F. (2024). Motivations, challenges, best practices, and benefits for bots and conversational agents in software engineering: A multivocal literature review. *ACM Computing Surveys*, *57*(4).

Liao, Z., Huang, X., Zhang, B., Wu, J., & Cheng, Y. (2023). BDGOA: A bot detection approach for GitHub OAuth apps. *Intelligent and Converged Networks*, *4*(3), 181–197.

Liao, Z., Qi, X., Zhang, Y., Fan, X., Zhou, Y., & Huang, C. (2020). How to evaluate the productivity of software ecosystem: A case study in GitHub. *Scientific Programming*, *2020*.

Lin, H. Y., Thongtanunam, P., Treude, C., & Charoenwet, W. (2024). Improving automated code reviews: Learning from experience. In *International Conference on Mining Software Repositories (MSR)*. ACM.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *International Conference on Neural Information Processing SystemsNeurIPS*, NIPS'17, (p. 4768–4777). ACM.

Ma, Y., Dey, T., Bogart, C., Amreen, S., Valiev, M., Tutko, A., Kennard, D., Zaretzki, R., & Mockus, A. (2021). World of code: enabling a research workflow for mining and analyzing the universe of open source VCS data. *Empirical Software Engineering (EMSE)*, *26*, 1–42.

Mann, H. B., & Whitney, D. R. (1940). On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, *11*(2), 147–162.

McClean, K., Greer, D., & Jurek-Loughrey, A. (2021). Social network analysis of open source software: A review and categorisation. *Information and Software Technology*, *130*.

Mienye, I. D., Swart, T. G., & Obaido, G. (2024). Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, *15*(9).

Mirhosseini, S., & Parnin, C. (2017). Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *International Conference on Automated Software Engineering (ASE)*, (pp. 84–94). IEEE/ACM.

Moharil, A., Orlov, D., Jameel, S., Trouwen, T., Cassee, N., & Serebrenik, A. (2022). Between JIRA and GitHub: ASFBot and its influence on human comments in issue trackers. In *International Conference on Mining Software Repositories (MSR)*, (pp. 112–116). IEEE Computer Society.

Mohayeji, H., Agaronian, A., Constantinou, E., Zannone, N., & Serebrenik, A. (2023). Investigating the resolution of vulnerable dependencies with Dependabot security updates. In *International Conference on Mining Software Repositories (MSR)*, (pp. 234–246). IEEE/ACM.

Monperrus, M., Urli, S., Durieux, T., Martinez, M., Baudry, B., & Seinturier, L. (2019). Repairnator patches programs automatically. *Ubiquity*, *2019*(July), 1–12.

Murali, A., Sahu, G., Thangarajah, K., Zimmerman, B., Rodríguez-Pérez, G., & Nagappan, M. (2024). Diversity in issue assignment: humans vs bots. *Empirical Software Engineering (EMSE)*, *29*(2).

Onsori Delicheh, H., Decan, A., & Tom, M. (2024). Quantifying security issues in reusable JavaScript Actions in GitHub workflows. In *International Conference on Mining Software Repositories (MSR)*, (p. 692–703). ACM.

Oriol, M., Müller, C., Marco, J., Fernandez, P., Franch, X., & Ruiz-Cortés, A. (2023). Comprehensive assessment of open source software ecosystem health. *Internet of Things*, *22*.

Orrei, V., Raglianti, M., Nagy, C., & Lanza, M. (2023). Contribution-based firing of developers? In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, (p. 2062–2066). ACM.

Rebai, S., Ben Sghaier, O., Alizadeh, V., Kessentini, M., & Chater, M. (2019). Interactive refactoring documentation bot. In *International Working Conference on Source Code Analysis and Manipulation (SCAM)*, (pp. 152–162). IEEE.

Rebatchi, H., Bissyandé, T. F., & Moha, N. (2024). Dependabot and security pull requests: large empirical study. *Empirical Software Engineering*, *29*(5).

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Gupta, B. B., Chen, X., & Wang, X. (2021). A survey of Deep Active Learning. *ACM Computing Surveys*, *54*(9).

Ribeiro, E., Nascimento, R., Steinmacher, I., Xavier, L., Gerosa, M., de Paula, H., & Wessel, M. (2022). Together or apart? Investigating a mediator bot to aggregate bot's comments on pull requests. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 434–438). IEEE.

Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., & Odenwald, T. (2009). Open collaboration within corporations using software forges. *IEEE Software*, *26*(2), 52–58.

Romano, J., Kromrey, J. D., Coraggio, J., Skowronek, J., & Devine, L. (2006). Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices. In *Annual Meeting of the Southern Association for Institutional Research*.

Romero, R., Parra, E., & Haiduc, S. (2020). Experiences building an answer bot for gitter. In *International workshop on Bots in Software Engineering (BotSE)*, (p. 66–70). ACM.

Rostami Mazrae, P., Mens, T., Golzadeh, M., & Decan, A. (2023). On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. *Empirical Software Engineering*, *28*(2).

Saadat, S., Colmenares, N., & Sukthankar, G. (2021). Do bots modify the workflow of GitHub teams? In *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 1–5). IEEE/ACM.

Safavian, S., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *Transactions on Systems, Man, and Cybernetics*, *21*(3), 660–674.

Schueller, W., Wachs, J., Servedio, V. D. P., Thurner, S., & Loreto, V. (2022). Evolving collaboration, dependencies, and use in the Rust open source software ecosystem. *Scientific Data*, *9*(1).

Snell, J., Swersky, K., & Zemel, R. (2017). Prototypical networks for few-shot learning. In *International Conference on Neural Information Processing SystemsNeurIPS*, NIPS'17, (p. 4080–4090). Red Hook, NY, USA: Curran Associates Inc.

Theil, H. (1967). Economics and information theory. *Amsterdam: North*.

Tsay, J., Dabbish, L., & Herbsleb, J. (2014). Let's Talk about It: Evaluating Contributions through Discussion in GitHub. In *International Symposium on Foundations of Software Engineering*, FSE 2014. ACM.

van Buuren, S. (2012). *Flexible Imputation of Missing Data*. Champan and Hall.

van de Ven, G. M., Tuytelaars, T., & Tolias, A. S. (2022). Three types of incremental learning. *Nature Machine Intelligence*, *4*(12), 1185–1197.

Van Ness, M., Bosschieter, T. M., Halpin-Gregorio, R., & Udell, M. (2023). The missing indicator method: From low to high dimensions. In *SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD, (pp. 5004–5015). ACM.

Vasilescu, B., Serebrenik, A., & van den Brand, M. (2011). You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics. In *International Conference on Software Maintenance (ICSM)*, (pp. 313–322). IEEE.

Wang, L., Zhang, X., Su, H., & Zhu, J. (2024). A comprehensive survey of continual learning: Theory, method and application. *Transactions on Pattern Analysis and Machine Intelligence*, *46*(8), 5362–5383.

Wang, Z., Wang, Y., & Redmiles, D. (2022). From specialized mechanics to project butlers: The usage of bots in open source software development. *IEEE Software*, *39*(5), 38–43.

Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Wessel, M., Abdellatif, A., Wiese, I., Conte, T., Shihab, E., Gerosa, M. A., & Steinmacher, I. (2022a). Bots for pull requests: The good, the bad, and the promising. In *International Conference on Software Engineering (ICSE)*, (pp. 274–286). IEEE/ACM.

Wessel, M., De Souza, B. M., Steinmacher, I., Wiese, I. S., Polato, I., Chaves, A. P., & Gerosa, M. A. (2018). The power of bots: Understanding bots in OSS projects. *International Conference on Human-Computer Interaction (CHI)*.

Wessel, M., Mens, T., Decan, A., & Mazrae, P. R. (2023a). *The GitHub Development Workflow Automation Ecosystems*, (pp. 183–214). Springer.

Wessel, M., Serebrenik, A., Wiese, I., Steinmacher, I., & Gerosa, M. A. (2022b). Quality gatekeepers: investigating the effects of code review bots on pull request activities. *Empirical Software Engineering*, *27*(5).

Wessel, M., Steinmacher, I., Wiese, I., & Gerosa, M. A. (2019). Should I stale or should I close? an analysis of a bot that closes abandoned issues and pull requests. In *International Workshop on Bots in Software Engineering (BotSE)*, (p. 38–42). IEEE.

Wessel, M., Wiese, I., Steinmacher, I., & Gerosa, M. A. (2021). Don't disturb me: Challenges of interacting with software bots on open source software projects. *International Conference on Human-Computer Interaction (CHI)*, *5*.

Wessel, M., Zaidman, A., Gerosa, M. A., & Steinmacher, I. (2023b). Guidelines for developing bots for GitHub. *IEEE Software*, *40*(3), 72–79.

Witten, I. H., & Frank, E. (2002). Data mining: practical machine learning tools and techniques with Java implementations. *SIGMOD Rec.*, *31*(1), 76–77.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer.

Wyrich, M., & Bogner, J. (2019). Towards an autonomous bot for automatic source code refactoring. In *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 24–28). IEEE/ACM.

Wyrich, M., Ghit, R., Haller, T., & Müller, C. (2021). Bots don't mind waiting, do they? Comparing the interaction with automatically and manually created pull requests. In *International Workshop on Bots in Software Engineering (BotSE)*, (pp. 6–10). IEEE/ACM.

Young, J.-G., Casari, A., McLaughlin, K., Trujillo, M. Z., Hébert-Dufresne, L., & Bagrow, J. P. (2021). Which contributions count? Analysis of attribution in open source. In *International Conference on Mining Software Repositories (MSR)*, (pp. 242–253). IEEE/ACM.

Zhang, X., Yu, Y., Gousios, G., & Rastogi, A. (2023). Pull request decisions explained: An empirical overview. *Transactions on Software Engineering*, *49*(2), 849–871.

Zhang, X., Yu, Y., Wang, T., Rastogi, A., & Wang, H. (2022). Pull request latency explained: An empirical overview. *Empirical Software Engineering (EMSE)*, *27*(6).