Discovering the topics of Continuous Integration Projects on GitHub

Lukas Ostrovskis[1]

Supervisor(s): Sebastian Proksch[1]**, Shujun Huang**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 28, 2023

Name of the student: Lukas Ostrovskis
Final project course: CSE3000 Research Project
Thesis committee: Sebastian Proksch, Shujun Huang, Fenia Aivaloglou

Abstract

Continuous Integration (CI) is a software development technique that enhances software quality and development efficiency, but its implementation usually depends on the project's context. This creates an opportunity for studying real-world CI projects on GitHub, focusing on their CI metrics and best practices. In this paper, we explore various methods to extract the topics from CI software projects on GitHub. This data can then be used to group projects and facilitate an in-depth analysis within specific contexts and application domains, such as CI build success rates in machine learning or React Native projects. We explore the definition of a software topic, as it shows significant granularity variations in related studies. We examine existing tools and other potential topic modeling approaches, compare varying types of textual data from GitHub that could be used as inputs for these tools, and report on interesting insights from initial trials with the developed tool. Our research led us to use GitHub topic labels as topic definitions due to their relevance and prior research focus. We also evaluated three topic extraction tools - LASCAD, a Multi-label Linear Regression classifier, and ChatGPT - incorporating the last two into our CI project mining tool. Additionally, we included two tool-independent approaches: GitHub's search function with the ability to filter repositories by topics and existing project topic labels. Lastly, to test the practicality of the tool, we mined 4899 public repositories and briefly investigated workflow metrics of projects grouped into six arbitrary topics.

1 Introduction

Continuous Integration (CI) is a software development practice that involves merging code changes into a shared repository on a frequent basis. This practice ensures that each change is tested and verified promptly, allowing developers to quickly identify and fix issues. CI can improve software quality, efficiency, and reduce the likelihood of errors [1]. While there are various approaches to implementing CI, it is challenging to provide a general guideline since effectiveness can depend heavily on the context of each project, as highlighted in previous studies [2, 3]. Therefore, there is a broad configuration of project settings where specific execution of CI could be analyzed, comparing the various compromises made to accommodate project execution and contextual factors.

Analyzing CI software projects can yield valuable insights into best practices for Continuous Integration. It enables a better understanding of effective CI implementation in diverse contexts and thus provides guidelines for maturing Continuous Integration processes. Further research in this domain can assist organizations in enhancing their context-dependent software development processes, resulting in improved software quality, increased efficiency, and reduced error risks.

This research focuses on open-source software projects that utilize CI on GitHub, a widely used hosting service for software development. With over 40 million public repositories[1], GitHub provides an ideal platform for large-scale data analysis. This study draws inspiration from previous research that extracted data from GitHub to analyze software engineering practices [4, 5, 6].

The main goal of this study is to cluster software projects according to their topics, enabling the analysis of the correlation between CI implementations and different topics, and the discovery of emerging best practices in various domains. To achieve this, the research intends to define relevant topics for CI analysis in open-source software projects and propose effective methods for categorizing projects on GitHub. Although software categorization has lately attracted significant attention from researchers, the focus on GitHub projects in this study enables the utilization of additional methods and approaches which can make the analysis more efficient.

More specifically, the main research question is: *"What data can be extracted from GitHub to effectively classify Continuous Integration projects by topic, and what CI practices emerge from these topic clusters?"* In order to provide a complete answer to the main question, the following subquestions have to be answered:

Q1: Which categories or domains should be used for classifying CI projects?

Q2: What approaches exist for categorizing GitHub projects based on their topics?

Q3: How does the inclusion of various extracted data impact the effectiveness of an external software classifier tool?

Q4: What is the relationship between certain project topics that are identified through extracted data and the underlying CI practices?

The structure of this paper is organized as follows. Section 2 provides a concise background and related work discussion, briefly presenting related work in CI, exploring the definition and characteristics of a software project topic, and examining previous studies on software project topic analysis and categorization. In Section 3, we present our methodology, which is organized around five main components aimed at addressing the research questions. Section 4 presents the data analysis and results, covering the answers to the subquestions and providing insights into the categorization of CI projects, GitHub project categorization approaches, the impact of extracted data on classifier effectiveness, and the relationship between a chosen subset of project topics and CI practices. Section 5 presents a discussion of the interesting observations and insights we gained about the problem during the research, going beyond the specific results we obtained. Following this, Section 6 discusses the reproducibility of the research and the ethical implication. Finally, the paper concludes with Section 7, summarizing the research findings, addressing limitations, and offering suggestions for future research.

2 Background and related work

This section provides a concise overview of the background and related research. The first group of studies focuses on CI implementations and emphasizes the arguments motivating our research. The next set of studies focuses on the do-

[1] Information retrieved using GitHub Search

mains and topics utilized for describing the context of software projects. The final set of works examines various existing tools for topic recommendation and extraction.

2.1 Continuous Integration implementation differences

Multiple studies have highlighted the complexities and difficulties associated with implementing CI and noted the absence of a universally applicable solution as a concern. The primary inspiration for our research is based on these preceding studies. As a result, a couple of them, that have analyzed CI implementations, are mentioned in the following paragraphs.

Elazhary et al. analyzed the strategies employed by organizations to implement CI practices and explored the advantages and drawbacks they experienced [2]. Their findings emphasize the significance of project context in the execution of CI. The authors focused on project type, testing strategy and CI infrastructure, which all varied greatly among the investigated organizations. They also further highlight the implications of these contextual factors and the absence of a universal solution in CI implementation for practitioners. The authors suggest prioritizing the CI methodologies that suit the project's workflow and deliver the necessary functionality.

A subsequent study [3] conducted a systematic literature review to perform an analysis of the differences in CI practices across the industry. Their conclusions align with Elazhary et al., reiterating the lack of a universal approach to CI implementation. Attempting to improve understanding of CI practices, they also offered a descriptive model for CI variations as a result of their research. Importantly, the authors noted the potential value of exploring the connection between contextual differences and CI in further research. They identified it as an opportunity to provide valuable insights in this field.

2.2 Granularity of software project topics

Various research has utilized different levels of specificity to characterize software projects. These definitions range from broad application areas, such as *web libraries and frameworks* or *software tools*, to more specific topic tags, offering insights into a project's function and content type (e.g., {*c,vim,api,text-editor*}). By analyzing the different papers, we can arrive at an informed choice regarding the suitable level of detail for classifying CI software projects.

Initial studies aiming to classify software often opted for wider topic categories. One of the pioneering works, MUDABlue [7], proposed a dataset of projects written in C and divided into 6 categories (*boardgame, compilers, database, editor, videoconversion, xterm*). Later efforts, such as LACT [8] and [9] offered new datasets of projects classified under SourceForge[2] categories, comprising 6 (*game, editor, database, terminal, e-mail, chat*) and 22 categories, respectively.

Recent studies used and proposed new datasets of projects hosted on GitHub. Although GitHub launched its own repository classification system - topic labels - in early 2017[3], the level of granularity in academic research remained variable. ClassifyHub [10], for instance, used the InformatiCup 2017[4] dataset, segregating GitHub repositories into 7 categories (*dev, hw, edu, docs, web, data, other*). A language-agnostic categorization tool, LASCAD [11], introduced a dataset classified into 6 categories (*machine learning, data visualization, game engine, web framework, text editor, web game*), built from GitHub Collections. Most recent research focusing on topic recommendations, however, leverages refined GitHub topic labels for model training. The authors of MNBN [12] and TopFilter [13], for example, used a dataset of repositories tagged with 154 different featured topics. Repologue [14] introduced a massive dataset of around 152K GitHub repositories annotated with 228 featured topics. Their work also includes meticulous topic processing, as these 228 topics are derived from an initial collection of over 29K user-defined topics, which were grouped into a set of 335 topics, and further trimmed to ensure a fair representation of each topic.

Efforts have also been made to consolidate different classification levels into comprehensive software classification taxonomies. GitRanking [15], for instance, is "a framework for creating a classification ranked into discrete levels based on how general or specific their meaning is". The authors collected 121K topic labels from GitHub, resulting in a compilation of 301 application domains linked to Wikidata for term disambiguation. The hierarchy contains topics that become increasingly specific as one moves down the list, starting from *'science'*, *'mathematics'*, and *'physics'*, and going down to *'image segmentation'* and *'convolutional neural networks'* at the lower levels. The authors of [16] utilized StackOverflow[5] post tags to build a large-scale software programming taxonomy, resulting in around 38K concepts and 68K relations in total (e.g. *"neural network"→"deep learning"→"word2vec"*). Lastly, the authors of [17] improved upon the previous work on topic recommendation [14] by supplementing the topic dataset with semantic relationships between them and encoding this data into a knowledge graph[6]. This graph contains 2234 relationships across 13 different types (is-a, is-based-on, works-with, etc.) between 863 community-curated GitHub topics.

2.3 Existing tools for topic extraction and recommendation

Several studies have focused on the automated categorization of software application domains. Various tools, utilizing distinct techniques, have been developed to undertake this task. Through an analysis of these tools, we can identify suitable ones for integration into our own research. This discussion presents an overview of different studies employing different techniques for extracting and recommending software project topics. These range from earlier contributions utilizing Latent Semantic Analysis, later works combining Latent Dirichlet Allocation with other approaches, to more recent research

[2]https://sourceforge.net/

[3]https://github.blog/2017-01-31-introducing-topics/

[4]https://github.com/informatiCup/informatiCup2017

[5]https://stackoverflow.com/

[6]https://www.ibm.com/topics/knowledge-graph

leveraging modern multi-label classifiers and transformers.

One of the earliest software categorization tools is the previously mentioned MUDABlue [7]. This model employs Latent Semantic Analysis (LSA) solely on a project's source code. It generates a set of concepts linked to the documents and terms and develops category clusters through the analysis of cosine similarities. Following MUDABlue, LACT [8] was developed, adopting an alternative Information Retrieval (IR) approach, Latent Dirichlet Analysis (LDA). It uses identifiers and comments in the source code to form category clusters with a cosine similarity exceeding 0.8.

Later tools built upon the potential of the IR technique LDA, merging it with other advanced techniques. For example, LDA-GA [18] combined LDA with Genetic Algorithms, improving the hyperparameters configuration for LDA and thus enhancing accuracy on the software labeling dataset. Additionally, LASCAD [11], a language-independent software categorization tool, merged LDA and hierarchical clustering. This eliminates the need to fine-tune the LDA parameter for determining the number of latent topics, reducing the manual efforts required by developers in parameter tuning. LASCAD is also one of the earliest tools evaluated using a dataset spanning projects in different programming languages, making it valuable for large-scale research in software project contexts, even though it only used source code as input.

The most recent research focuses not only on topic extraction but also on recommendations based on the expanding set of GitHub topic labels. MNBN's [12] authors created a probabilistic model named Multinomial Naive Bayesian Network (MNBN) to suggest new relevant featured topic labels for a project by using README files and source code encoded via TFIDF. Repologue [14] assessed several methods, ranging from traditional multi-label classifiers to advanced methods like FastText (a library developed by Facebook for sentence classification) [19] and a transformer model, DistilBERT [20]. Notably, among the evaluated methods, a linear regression model (one of the traditional classifiers) was among the 2 best-performing methods, demonstrating evaluation scores comparable to the state-of-the-art transformer model, DistilBERT.

3 Methodology

The methodology for this research was organized around five main components, each designed to address the research questions stated in the introduction. The components are illustrated in Figure 1 below.

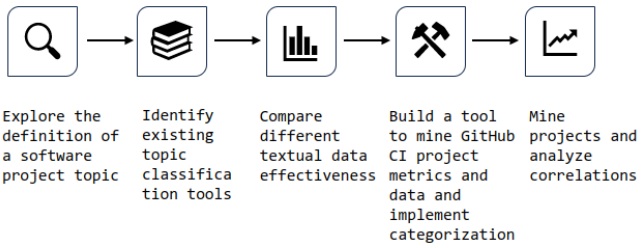

Figure 1: The 5 stage methodology of this research

The initial step was conducting a comprehensive literature review, targeted at resolving questions Q1 and Q2. After possible approaches for categorizing software projects based on their topics were identified, chosen tools were analyzed further and evaluated using diverse data extracted from GitHub, which ultimately helps answer Q3. Finally, the selected categorization methods were incorporated within our CI project mining tool, allowing us to cluster the examined projects according to their topics. This facilitated the analysis of correlations between various project topic domains and the emerging details of their CI implementations.

The literature review was focused on two main aspects. Firstly, we explored the definition of a software project topic by examining existing works that have attempted to create a coherent set of topics or even topic taxonomies related to software projects. The goal was to gain an understanding of the relevant topics to consider during the analysis of CI projects and determine the most suitable representation in this particular context. In the second part of the review we investigated existing topic classification tools in the field of software categorization. The objective was to select a small subset of tools that could be further analyzed and potentially integrated into our own tool.

To select an appropriate classification tool, we evaluated different software categorization tools based on their performance with varying textual data extracted from software projects. The comparative analysis considered criteria such as accuracy, clustering evaluation metrics such as macro-precision and macro-recall, processing time, and ease of integration into our existing tool. This analysis was particularly important as the nature and volume of the data might significantly influence the performance of the selected tool. Comparative analysis tables were prepared, mapping the performance metrics of each tool against our criteria, thus enabling an informed decision.

Following the decisions related to the relevant topics, suitable categorization tools, and the optimal textual data to extract from software projects, we integrated the chosen solutions into our existing mining tool[7]. This integration allowed for the clustering of the mined CI software projects based on their respective topics, thus establishing a foundation for the subsequent analysis of the clustered CI implementations in various application domains and contexts.

Finally, an initial exploration was conducted to investigate differences between CI utilization in projects across a set of topics. While a comprehensive correlation analysis is a time-consuming effort and was mainly deferred to future research, a carefully selected set of topics, manageable for the current analysis, was examined. The focus was placed on analyzing CI metrics within the context of these chosen topics.

4 Data Analysis and Results

In this section, conclusions drawn from the literature review and the experimental setup are presented, focusing on the complex aspects of topic modeling for CI software projects on GitHub. As defined in the methodology, the initial phase involves discussing the results of the literature review, where

[7]https://github.com/raduConstantinescu/Descriptive-CI-Metrics

insight is provided into the definition of a software project topic and existing methodologies for topic extraction, thereby addressing research questions Q1 and Q2. Subsequently, research question Q3 is addressed by evaluating the performance of various models, identified in the previous phase, using different textual data obtained from software projects. The evaluation is performed based on several metrics including Accuracy, Macro-Recall, Macro-Precision, Macro-F1 score, and processing time. Finally, research question Q4 is approached by investigating differences between a set of 6 arbitrary software project topics and their CI metrics.

Definition of a software topic (Q1)

As observed in related work, the definition of a software project topic is a matter of ongoing debate due to the absence of a general consensus on an appropriate granularity level. Consequently, previous research has aimed to formulate software topic taxonomies, yet these efforts were predominantly focused on the establishment of a coherent hierarchy rather than developing a framework to model software project topics based on a given set of inputs [16, 17]. The complexity and time constraints related to training a model based on these hierarchies, coupled with potential compromises on model performance, render this approach not entirely suitable for the purposes of our study. On the other hand, the scope of topic definitions in the work focused specifically on topic extraction frameworks varies greatly, ranging from broad application domains [10, 11] to highly specific topic labels [14]. We will thus discuss this definition span and explain the choice we deem appropriate in the context of this research.

Firstly, some of the broader domains, in ClassifyHub [10], for instance, such as *dev*, *hw*, and *docs*, are hardly useful due to insufficient contextual information since the majority of the software repositories with CI would be categorized into the *dev* category, rendering the categorization useless. Furthermore, while SourceForge categories, which are the base of multiple early works [8, 9] provide a viable option, their application to GitHub projects - lacking native SourceForge categorization - may not be easily achievable, and could thus lead to limited training accuracy. The importance of precision cannot be overstated in the current study context, where the role of topic modeling is not evaluated in isolation but rather as an instrumental tool integrated into the broader research. Any inaccuracies resulting from suboptimal selections could jeopardize the validity of the overall research findings.

Secondly, as shown in multiple studies, the role and application of CI can be highly context-dependent [3]. For instance, it is not only the implementation of CI in *mobile apps* that could be of interest to a reader but also the specific frameworks employed, such as *React Native* or *Flutter*. Consequently, a more nuanced and multifaceted approach to topic labeling may be beneficial.

Therefore, considering the primary focus of this study - mining GitHub projects - the topic labels provided by GitHub offer a practical solution. A notable concern with this approach, however, lies in the customizable nature of these labels. Users have the ability to create unique labels, leading to the existence of over 1M distinct topic labels employed to tag repositories on the platform[8]. Nevertheless, several points suggest that this challenge can be effectively conquered:

1. GitHub maintains a list of topic labels that have been curated by the community[9]. While the number of these labels remains substantial (exceeding 850), further analysis revealed a power-law distribution, illustrated in Figure 2, whereby approximately 80% of repositories linked to these curated labels use only about 20% of them. This effectively reduces the number of labels associated with a significant number of repositories to under 200.
2. The authors of Repologue [14] managed to condense the GitHub labels into a more manageable set of 228. The reduction process not only relied on statistical significance but also included manual, human-assisted topic mapping to generate a coherent and granular set of topics that are as concise as possible. Additionally, they provided extensive models and training datasets for extracting and recommending topic labels, establishing a promising foundation for our research.

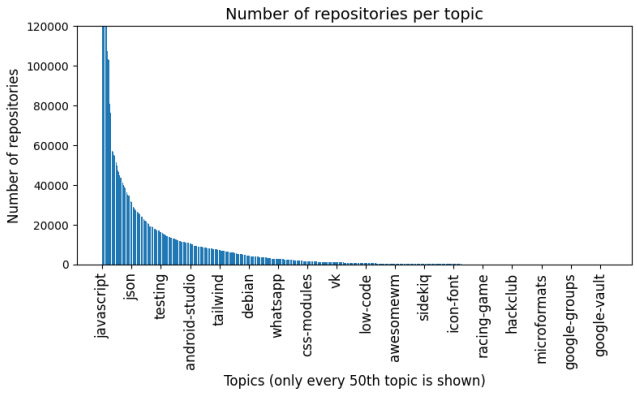

Figure 2: Power law-like distribution of repositories associated with GitHub community-curated topic labels. The x-axis displays a subset of topics (every 50th topic), as the full dataset comprises over 850 labels.

Consequently, despite the complexity and diversity of GitHub topic labels, prior research methods and the intrinsic distribution characteristics of the labels suggest that they can be effectively used in our study. This indicates the feasibility of employing GitHub topic labels for contextual-dependent CI project research, providing a promising approach for further exploration.

Tools and approaches (Q2)

As outlined in Section 2, numerous frameworks have been developed to facilitate the clustering of software projects based on topics. In the majority of instances, these frameworks are used together with topic categories of author-determined granularity. It is essential to emphasize that our

[8]Information retrieved using GitHub Search

[9]https://github.com/github/explore

decision on the appropriate topic definition may also considerably influence and even limit the usability of existing tools. However, the utilization of GitHub labels as topics for clustering repositories introduces approaches independent of external topic modeling tools, thereby reducing the potential for incorrect project categorization. The subsequent paragraphs explore these approaches and the relevant existing topic modeling tools highlighted in Section 2.

Firstly, the adoption of GitHub labels enables the utilization of the GitHub search function to conveniently retrieve projects of specific topics. While this approach restricts the capacity to mine arbitrary repositories, it proves valuable when specific topics are of interest. The tool users can easily compile a collection of GitHub repositories labeled with the selected topic. This strategy simplifies the task of gathering a statistically meaningful quantity of repositories related to a particular topic, concurrently minimizing the chances of incorrect categorizations.

Secondly, pre-existing topic labels can be exploited for clustering. Contrasting with the GitHub search, this strategy is unrestrictive, allowing the user to mine random repositories and offering an optional use of their pre-existing labels for aggregate analysis. Notably, doubts may arise regarding the efficacy of this approach given that the majority of projects on GitHub lack topic labels. Nevertheless, we hypothesized that CI projects are more likely to feature at least one topic label compared to non-CI projects, adding value to this approach. In a study of 4109 repositories, it was observed that 421 out of 2499 (24.24%) non-CI repositories had at least one topic label, while 1029 out of 2299 (50.49%) CI repositories had at least one. The null hypothesis was consequently rejected with a p-value of 0.01, applying the chi-squared test.

Lastly, the generation of topic labels for clustering can be achieved through existing topic modeling tools. While these tools are essential for the remaining projects that lack topic labels, they can also be utilized to enhance the topic set for each mined project, although with the risk of incorrect categorizations. The tools identified through the literature review for further analysis include:

- LASCAD [11]: this tool, based on Latent Dirichlet Allocation (LDA) and hierarchical clustering, is language-agnostic, making it suitable for categorizing projects of any language and thus expanding the scope of our research.
- Multi-label Linear Regression Classifier [14]: a promising tool based on GitHub topic labels with a large dataset[10] and encouraging performance.
- ChatGPT: although requiring payment for the use of API, this state-of-the-art technology offers a quick setup without the necessity for training. It represents a benchmark against which tools specifically designed for topic modeling can be compared.

Several other tools were not considered for further analysis due to various reasons such as having a narrowly focused research scope (e.g., focusing solely on Java projects), employing SourceForge categories (which were dismissed as outlined in the preceding subsection), or exhibiting poorer performance relative to the selected approaches. All the aforementioned tools were evaluated on the LASCAD [11] dataset of 103 software projects with a diverse configuration of textual data. The best results achieved by these tools (regardless of the textual data used) are presented in Table 1.

Table 1: Comparison of 3 tools used for software categorization by topics with a dataset of 103 GitHub repositories

	LASCAD	Multi-label LR Classifier	ChatGPT
Accuracy	0.72	0.57	0.95
Macro-Precision	0.72	0.78	0.95
Macro-Recall	0.62	0.62	0.92
Macro-F1 score	0.67	0.69	0.94
Processing time	2079.1	11.9	186.4

The table illustrates that the large language model ChatGPT outperforms other tools, achieving an impressive accuracy of 0.95 in classifying projects into six categories. However, further testing with a broader range of topics would be necessary to verify whether it is generally applicable. The second-best tool, LASCAD, had longer processing times, mainly when using source code as input. However, as seen in Table 4, LASCAD performed worse than the multi-label classifier in the majority of the metrics with all other subsets of textual data. Notably, the realistic accuracy for specialized topic-modeling tools seems to be below 70%, so we have to be cautious about overutilizing them in further research.

Impact of various extracted data on the effectiveness of a categorization tool (Q3)

GitHub repositories encapsulate a wide range of information beyond source code, including, such as README files, wiki pages, repository names and descriptions, and filenames along with their extensions. Previous research has already leveraged these diverse data types for topic extraction and even discussed their necessity for topic modeling [14]. However, in addition to accuracy-focused categorization metrics, the total processing time for topic extraction - from data collection to topic modeling - is of crucial importance for our tool. Given the multimodal nature of our tool, it is necessary to ensure that topic modeling does not induce a bottleneck and thus constrain the capabilities of CI project analysis.

We also hypothesize that the textual data, excluding source code, from CI projects could potentially be more informative on average than that from non-CI projects. The reasoning behind this hypothesis is that developers who have invested effort into setting up a CI pipeline are likely to also put in more effort in the proper project documentation. Consequently, this could enhance the performance of the topic extraction tools, leading to more accurate results. To validate this hypothesis,

[10] https://drive.google.com/drive/folders/1HdY3ykFSRdIqv91Ej6w0e-5Pov3Cg0O4

we mined data from 4109 arbitrary projects and assessed the length of the README files in these repositories. Our findings revealed a statistically significant difference (p = 0.01) between the average README file length for CI software repositories (5012.99) and those without CI (2590.14), leading us to reject the null hypothesis. Therefore, we decided to conduct our evaluation of the tools selected in the prior subsection using the same set of projects, utilizing different combinations of textual data retrieved from each repository.

We first identified the potential sources of textual data for categorization:

- Source code
- Repository name and description
- README files
- Commit messages
- Pull request titles and bodies
- Issue titles and bodies

It is important to acknowledge two significant constraints when retrieving data from GitHub repositories via the provided API: retrieval time and the number of API requests required to obtain the necessary data. GitHub provides 5000 API requests per hour, strongly limiting the volume of data that can be mined, thus affecting the types of data appropriate for mining in the context of our tool. After extracting data from the same 103 repositories used in the LASCAD study [11], we computed the average retrieval time and the average number of requests per repository for each data type. These results are summarized in Table 2.

Table 2: Comparison of different textual data mining from GitHub repositories, in terms of retrieval time and the number of used API requests

	Source code	Repository Name and Description	README
Avg. Retrieval Time (s)	89.5	1.2×10^{-5}	0.23
Avg. # of used API requests	0	1	1
	Commits	Pull Requests	Issues
Avg. Retrieval Time (s)	89.2	98.4	115.6
Avg. # of used API requests	406.8	76.9	164.8

As evident in the table, the most resource-intensive data types include source code, commits, pull requests, and issues, with an average retrieval time per instance approximating or exceeding 90 seconds. Furthermore, the API requests across these data sources exceed 600 requests per repository, which would restrict us to mining an average of approximately 8 repositories per hour. Given that this would be a severe bottleneck to future empirical research requiring large volumes of observed data, we propose to restrict the data used for topic modeling to README files, along with the repository name and description. Nevertheless, in the interests of comprehensiveness, we have assessed the impact of each of these data types on the chosen tools and will present it as well.

After determining the potential data types for extraction, we conducted an evaluation using three selected tools on the same dataset of 103 repositories. The metrics employed for this evaluation included Accuracy, Macro-Precision, Macro-Recall, and Macro-F1 score, commonly used in multiclass clustering performance analyses. Moreover, we included processing time as an evaluation metric, given its importance to our study. Tables 4 and 5 present the results of the evaluation for the LASCAD tool [11] and the Multi-label Linear Regression Classifier [14], and ChatGPT, respectively.

Analysis of CI metrics in projects with a set of hand-picked topics (Q4)

To evaluate the practicality of our newly developed CI project mining tool, we collected data from 4899 public repositories utilizing GitHub Action workflows[11]. Our investigation was focused on six arbitrary topics: Application Programming Interface (API), Android, iOS, JavaScript, TypeScript, and Docker. After identifying projects related to these topics, we also mined over 112K workflow jobs executed in those projects. Our goal was to aggregate the workflow metrics across projects within these topics, calculating the average workflow count per project, the mean number of workflow runs triggered by both pull request submissions and direct code pushes to the repositories, and the success rate of workflows' jobs. The results are presented in Table 3.

Table 3: GitHub Actions workflow metrics of repositories with different topics

	api	*android*	*ios*
Projects with workflows (%)	64.1	59.8	53.1
Avg. workflow count	1.88	0.82	0.65
Avg. pull request runs	1437	215.6	7.4
Avg. push runs	586.4	63.3	6.7
Job success rate	0.81	0.83	0.69
	javascript	*typescript*	*docker*
Projects with workflows (%)	56.1	73.8	80
Avg. workflow count	1	1.54	1.04
Avg. pull request runs	179	121.6	30.4
Avg. push runs	148	127	60.9
Job success rate	0.81	0.81	0.93

The findings offer valuable insights into the utilization of

[11]https://docs.github.com/en/actions/using-workflows/about-workflows

workflows across diverse project categories. For example, Docker and TypeScript projects are more inclined toward the use of workflows. Notably, API projects exhibit the highest average workflow count per repository, indicating that developers within this domain find value in combining multiple workflows. API projects also utilize the workflows significantly more than other categories, with roughly six times more average workflow runs than the next most active topic, JavaScript. This could imply a higher level of complexity in API projects, where frequent workflow runs are necessary to avoid critical problems.

An interesting anomaly is observed in the triggering of workflow runs. While most projects primarily initiate workflow runs upon pull request submissions, Docker projects contradict this trend by favoring code push triggers, as evidenced by the 2:1 ratio. Lastly, the data reveals an increased inclination towards workflow use in TypeScript projects compared to JavaScript ones. However, TypeScript projects show fewer average workflow runs, possibly due to TypeScript's inherent type safety, which may reduce the need for external validation.

5 Discussion

In this section, we discuss new insights and our deeper understanding of the research problem, that we think are particularly interesting and could influence future research direction and methodologies.

One of the significant challenges we encountered is the issue of software topic granularity, which varies greatly across works related to topic modeling. Existing research has mostly focused on the generation of software domain hierarchies and the development of topic extraction tools as separate challenges. Although there are recent efforts to incorporate semantic relationships into topic modeling to boost performance, no evident attempts have been made to integrate easily configurable hierarchies into such models.

Moreover, topic modeling has seen significant advancements over the years, with research employing increasingly sophisticated techniques. However, the accuracy of such methods is not perfect. It is important to be cautious when utilizing these tools in broader research, as inconsistencies in the extraction of topics may skew the results. Future research could explore how different topic modeling tools impact the outcomes of controlled, context-dependent empirical analyses.

Our research also showcased the promising capabilities of large language models (LLMs) in topic modeling, although our experiments were limited to only 6 topics for classification. We remain uncertain about how LLMs, such as ChatGPT, would perform with larger lists of topics. However, the massive datasets that these models are trained on could be the main advantage over specialized topic modeling tools. Therefore, given the rapid advancement of LLMs, this is an approach that should be considered in further research.

Finally, while our research focused on mining random projects and extracting their topics for empirical CI analysis, we argue that leveraging GitHub's search function could be a useful strategy for investigating specific topics. For instance, in random analysis, only a small percentage of total mined projects will belong to a specific topic. Conversely, GitHub Search allows users to create a list of projects related to a single topic, enabling more efficient results when investigating specific domains. This approach also reduces the risk of incorrect classifications by external topic modeling tools, thus minimizing threats to the validity of future work.

6 Responsible Research

Conducting research in a responsible manner is critical, especially when the results are expected to contribute significantly to the field of study. The process of mining CI software projects from GitHub to model their topics inherently implies potential ethical and reproducibility challenges, which are discussed in the following paragraphs.

Firstly, the act of mining repository data from GitHub, which is a form of data mining, elicits a few ethical concerns. The exploration of the ethics of software repository mining is covered in great detail in [21], with emphasis on numerous ethical issues that should be considered. One particularly relevant issue is the "Identification of Stakeholders and Informed Consent". Stakeholders comprise individuals, entities, or institutions that might have an interest in being informed about the data collection process. In the context of our study, the primary stakeholders would be GitHub and the developers involved in the software repositories that were mined as part of this research. GitHub's "Acceptable Use Policies"[12] explicitly permits the usage of public information for research purposes, provided the resulting research is available via open access. Additionally, the repositories mined for this study were open-source, which we deem a conscious decision made by the repository maintainers to share their projects with the public. These factors, combined with the stated policy of GitHub, reasonably support our conclusion that the act of mining repositories for this research is not inherently unethical.

Secondly, there exists an ethical dimension in relation to the use of topic modeling as a building block in the research of CI projects. As shown in previous sections, topic extraction tools are known to be imperfect in terms of accuracy, and therefore their outputs should ideally be treated as suggestive rather than being considered as the ground truth. This could potentially compromise the validity of future research if the limitations of these tools are not properly evaluated and discussed. To address this, we aimed to minimize reliance on these tools in general, opting instead to use GitHub's native topic labels whenever feasible and making the use of topic modeling tools optional and user-adjustable. Furthermore, we have tried to maintain transparency regarding the accuracy of these tools to enable readers to draw their own conclusions on this matter.

Lastly, in order to follow the principles of research reproducibility and transparency, only original experimental data has been included in this paper. We have ensured that all used and generated datasets and models, as well as the developed CI project mining tool, are accessible to the reader.

[12] https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies

The methodology employed in the study is described in a clear, step-by-step manner to enable readers to replicate the research process and reproduce the results using the provided code and datasets.

7 Conclusions

This study aimed to answer the question *"What data can be extracted from GitHub to effectively classify Continuous Integration (CI) projects by topic, and what CI practices emerge from these topic clusters?"*. Our research focused on exploring the best approaches for categorizing CI projects by topic, evaluating topic modeling tools, analyzing the impact of different extracted data on external software classifiers, and examining the correlation between identified project topics and underlying CI practices.

Our findings led us to use GitHub's topic labels, as given their relevance, support for, besides external tools, additional repository clustering methods, and prior focus in research, they prove to be hugely convenient. Furthermore, regarding the approaches for categorizing GitHub projects, three tools were evaluated - LASCAD [11], a Multi-label Linear Regression classifier [14], and ChatGPT. As we were satisfied with the balance between their clustering performance and processing time, the latter two were incorporated into our own CI project mining tool.

With respect to the type of data extracted from GitHub, we discovered that the combination "repository name, description, and README" offer the best balance between classifier performance and processing time. Conversely, the combination "commits, issues, and pull requests" demonstrated disappointing performance. Source code, while useful with appropriate preprocessing, significantly prolonged processing time, which could limit the scale of future research.

Lastly, after developing our CI project mining tool[13], we briefly examined data from 4899 public repositories using GitHub Action workflows across six topics, aiming to test and showcase the usability of our developed tool. Despite their limited scope, our results, as shown in Table 3, offer interesting insights into the differences in CI utilization across different contexts. For instance, we observed a higher workflow count in API projects and a deviation from the standard trend in Docker projects, where workflow triggers due to code pushes surpassed those from pull requests. Thus, we believe that our developed tool can be successfully employed in future research focusing on context-dependent CI implementations, further exploring the variation of CI practices across diverse software domains and methodologies. This could, consequently, improve our understanding of CI optimization across various development scenarios.

7.1 Limitations

This study was bounded by certain constraints, primarily time, which led to several limitations affecting its scope and depth.

One of the limitations was the evaluation of only a small subset of existing topic modeling tools. We did not have the capacity to assess all of the recent state-of-the-art tools, hence our selection might not represent the truly optimal tool for topic discovery of CI projects. With an extended timeline, a broader exploration of related tools could be conducted, potentially leading to different results and revealing the most effective tools.

The time constraint also impacted the size and breadth of the dataset that the tools were evaluated on. The relatively small set of 103 projects clustered into 6 different topics could undermine our findings' generalizability. A more extensive dataset could have increased the validity of our performance evaluations and allowed for deeper understanding of how different tools perform across a broader range of both projects and topics.

7.2 Future Work

This study paves the way for a variety of opportunities related to empirical analysis of context-dependent CI implementations. Subsequent research can focus either on improving our developed CI software project mining tool or utilizing it for large-scale empirical analysis.

Firstly, comparison of existing topic modeling tools could be improved. This can be achieved by increasing the number of compared projects and including a broader range of tools, even those initially discarded in this study. Retraining models on the actual subsets of utilized textual data, instead of relying solely on models pre-trained on a predetermined subset of data, may offer further improvements in performance.

In addition to improving tool comparison, extending the performance evaluation to compare the differences in effectiveness of textual data in CI and non-CI projects could provide valuable insights. This could uncover useful correlations between CI usage and the informativeness of project's textual data, leading to more informed decisions regarding the most effective data subsets for CI research.

Exploring the integration of semantic relationships into topic modeling is another promising direction. The introduction of topic hierarchies into CI analysis in software projects could provide a better understanding of the influence and interactions between different topics and CI practices. It would also simplify the process of analysis, since easily configurable hierarchical models would bring flexibility, allowing topic predictions to be as detailed or as broad as needed, suiting the unique requirements of each study.

Finally, employing the mining tool developed in this study to analyze a wide range of projects across different contexts could uncover new insights into Continuous Integration. This extensive analysis has the potential to form concrete guidelines for future CI implementation strategies and improvements.

[13] https://github.com/raduConstantinescu/Descriptive-CI-Metrics

A Textual data and topic modeling tool evaluation tables

Table 4: Comparison of different textual data from GitHub repositories using LASCAD and a Multi-label Linear Regression Classifier (MLRC) for clustering repositories by topics

	Source code		README		Repository Name, Description, README		Commits, Issues, Pull Requests		All but source code		All	
	LASCAD	MLRC	LASCAD	MLRC	LASCAD	MLRC	LASCAD	MLRC	LASCAD	MLRC	LASCAD	MLRC
Accuracy	0.72	0.4	0.46	0.52	0.53	0.57	0.2	0.19	0.54	0.52	0.71	0.42
Macro-Precision	0.72	0.59	0.63	0.76	0.56	0.78	0.3	0.19	0.62	0.76	0.77	0.62
Macro-Recall	0.62	0.45	0.4	0.57	0.47	0.62	0.17	0.25	0.52	0.57	0.64	0.48
Macro-F1 score	0.67	0.51	0.49	0.65	0.51	0.69	0.22	0.22	0.56	0.57	0.7	0.54
Time (s) for 103 repos	2079.1	3586.17	46.3	11.2	46.8	11.9	14.2	5.5	54.5	12.7	2132.4	3613.3

Table 5: Comparison of different textual data from GitHub repositories using ChatGPT for clustering repositories by topics

	README	Repository Name, Description	All
Accuracy	0.94	0.94	0.95
Macro-Precision	0.94	0.96	0.95
Macro-Recall	0.9	0.89	0.92
Macro-F1 score	0.92	0.92	0.94
Time (s) for 103 repos	183.7	167.2	186.4

References

[1] Bogdan Vasilescu et al. “Quality and productivity outcomes relating to continuous integration in GitHub”. In: *Proceedings of the 2015 10th joint meeting on foundations of software engineering*. 2015, pp. 805–816.

[2] Omar Elazhary et al. “Uncovering the benefits and challenges of continuous integration practices”. In: *IEEE Transactions on Software Engineering* 48.7 (2021), pp. 2570–2583.

[3] Daniel Ståhl and Jan Bosch. “Modeling continuous integration practice differences in industry software development”. In: *Journal of Systems and Software* 87 (2014), pp. 48–59.

[4] Chris Brown and Chris Parnin. “Understanding the Impact of GitHub Suggested Changes on Recommendations between Developers”. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1065–1076. ISBN: 9781450370431. DOI: 10.1145/3368089.3409722.

[5] Daye Nam, Youn Kyu Lee, and Nenad Medvidovic. “EVA: A Tool for Visualizing Software Architectural Evolution”. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 53–56. ISBN: 9781450356633. DOI: 10.1145/3183440.3183490.

[6] Baishakhi Ray et al. “A Large Scale Study of Programming Languages and Code Quality in Github”. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 155–165. ISBN: 9781450330565. DOI: 10.1145/2635868.2635922.

[7] S. Kawaguchi et al. “MUDABlue: an automatic categorization system for open source repositories”. In: *11th Asia-Pacific Software Engineering Conference*. 2004, pp. 184–193. DOI: 10.1109/APSEC.2004.69.

[8] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. “Using Latent Dirichlet Allocation for automatic categorization of software”. In: *2009 6th IEEE International Working Conference on Mining Software Repositories*. 2009, pp. 163–166. DOI: 10.1109/MSR.2009.5069496.

[9] Mario Linares-Vásquez et al. “On using machine learning to automatically classify software applications into domain categories”. In: *Empir. Software Eng.* 19.3 (June 2014), pp. 582–618. ISSN: 1573-7616. DOI: 10.1007/s10664-012-9230-z.

[10] Marcus Soll and Malte Vosgerau. “ClassifyHub: An Algorithm to Classify GitHub Repositories”. In: Sept. 2017, pp. 373–379. ISBN: 978-3-319-67189-5. DOI: 10.1007/978-3-319-67190-1_34.

[11] Doaa Altarawy et al. “Lascad : Language-agnostic software categorization and similar application detection”. In: *Journal of Systems and Software* 142 (2018), pp. 21–34. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.04.018.

[12] Claudio Di Sipio et al. “A Multinomial Naıve Bayesian (MNB) Network to Automatically Recommend Topics for GitHub Repositories”. In: *ResearchGate* (Apr. 2020), pp. 71–80. DOI: 10.1145/3383219.3383227.

[13] Juri Di Rocco et al. “TopFilter: An Approach to Recommend Relevant GitHub Topics”. In: *ResearchGate* (Sept. 2020). DOI: 10.1145/3382494.3410690.

[14] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. “Topic recommendation for software repositories using multi-label classification algorithms”. In: *Empir. Software Eng.* 26.5 (Sept. 2021), pp. 1–33. ISSN: 1573-7616. DOI: 10.1007/s10664-021-09976-2.

[15] Cezar Sas et al. “GitRanking: A Ranking of GitHub Topics for Software Classification using Active Sampling”. In: *ResearchGate* (May 2022). DOI: 10.48550/arXiv.2205.09379.

[16] Jiangang Zhu et al. “Building a Large-scale Software Programming Taxonomy from Stackoverflow”. In: *ResearchGate* (July 2015), pp. 391–396. DOI: 10.18293/SEKE2015-135.

[17] Maliheh Izadi, Mahtab Nejati, and Abbas Heydarnoori. “Semantically-enhanced Topic Recommendation System for Software Projects”. In: *arXiv* (May 2022). DOI: 10.48550/arXiv.2206.00085. eprint: 2206.00085.

[18] Annibale Panichella et al. “How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms”. In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 522–531. DOI: 10 . 1109 / ICSE . 2013 . 6606598.

[19] Armand Joulin et al. “Bag of Tricks for Efficient Text Classification”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 427–431.

[20] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv* (Oct. 2019). DOI: 10.48550/arXiv.1910.01108. eprint: 1910.01108.

[21] Nicolas E. Gold and Jens Krinke. “Ethics in the mining of software repositories”. In: *Empir. Software Eng.* 27.1 (Jan. 2022), pp. 1–49. ISSN: 1573-7616. DOI: 10.1007/s10664-021-10057-7.