Automatic Classification of Software Repositories: a Systematic Mapping Study

Stefano Balla
stefano.balla2@unibo.it
DISI - Università di Bologna
Bologna, Italy

Thomas Degueule
thomas.degueule@labri.fr
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800
Bordeaux, France

Romain Robbes
romain.robbes@labri.fr
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800
Bordeaux, France

Jean-Rémy Falleri
falleri@labri.fr
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Institut Universitaire de France
Bordeaux, France

Stefano Zacchiroli
stefano.zacchiroli@telecom-paris.fr
LTCI, Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France

Abstract

The rapid growth of software repositories on development platforms such as GitHub, as well as archives like Software Heritage, prompts the need for better repository classification. Machine learning is increasingly used to automate this classification, but there are no secondary studies analyzing this research landscape.

We present a systematic mapping study of 43 primary sources published between 2002 and 2023, where we examine the goals, inputs, outputs, training, and evaluation processes involved in automatic repository classification.

Our findings reveal a growing interest in automatic classification, particularly to enhance the discoverability and recommendation of relevant repositories. Other applications, such as classification for mining studies, were surprisingly underrepresented. We also observe that a lack of standardized datasets, classification tasks, and evaluation metrics makes it difficult to compare the performance of different techniques.

CCS Concepts

• **General and reference** → **Surveys and overviews**; • **Software and its engineering** → **Software libraries and repositories**; • **Information systems** → **Clustering and classification**.

Keywords

software repositories, repository classification, systematic mapping study

1 Introduction

Software engineering (SE) has seen rapid growth in the volume of version control system (VCS) repositories available (just "repositories", for short, in the following), particularly with collaborative commercial development platforms such as GitHub and GitLab, as well as academic platforms such as Software Heritage [7] and World of Code [14]. The sheer quantity of software repositories available—in excess of 345 million on Software Heritage alone, at the time of writing[1]—makes *automatic* repository classification highly desirable. This need is compounded by the complexity of modern software systems and the variety of classifications required to support various development and maintenance activities. Classification efforts in software repositories often aim to improve the efficiency of project management, code reuse, and collaboration, among other objectives [S32, S42, S50].

To classify software repositories, researchers tend to use automated techniques, particularly machine learning (ML). These techniques are used in various areas such as project tagging, repository recommendations, and code categorization. However, the landscape of repository classification remains fragmented, with various approaches, goals, and methodologies being employed. Researchers have developed numerous models for classification, yet little work has been done to comprehensively map the field and synthesize the results systematically. To fill this knowledge gap, we conducted a systematic mapping study (SMS) to provide an overview of current trends and identify areas where further research is required.

Paper structure. Section 2 details the methodology of our study. We focus on fundamental research questions, such as where and when the work was conducted, and the goals and motivations behind the classification efforts. Since current repository classification relies heavily on machine learning, we examine both data-related and algorithmic aspects. To ensure completeness and rigor, we followed a systematic methodology with defined inclusion/exclusion criteria, screening steps, and backward snowballing, validated through inter-rater reliability (Fleiss' kappa [9]) and refined via discussion. This led to the final selection of 43 studies, which were then categorized based on various dimensions related to our research questions. Section 3 presents our main results, which can be summarized as:

- There is a *long-running research interest* in repository classification, lasting more than two decades, with a spike in the last 7 years.
- The main stated motivation for repository classification is *discoverability*; surprisingly few studies address other themes.

[1] https://www.softwareheritage.org, visited on 2025-01-24

- A wide *variety of datasets, input data, features and metrics* are used, highlighting the richness and complexity of software repository data.

In Section 4, we examine the implications of our findings, including the strong emphasis on discoverability as the primary motivation, the absence of standardized benchmarks for datasets and evaluation methods, and the scalability challenges in applying these approaches to ultra-large-scale repository collections. Before concluding (Section 6), we discuss study limitations in Section 5.

Contributions and Data Availability. This paper presents (i) the first secondary study on the automatic classification of software repositories, and (ii) a publicly available replication package [1] containing the dataset of 43 analyzed papers and a structured taxonomy categorizing their key characteristics. All code and data used in this study are included in the package.

2 Methodology

To carry out this study, we followed the methodology proposed by Petersen et al. [17, 18], which was designed to ensure a comprehensive and reproducible process for selecting and analyzing relevant articles. The methodology is organized around three main phases: developing research questions (Section 2.1), selection of primary sources (Section 2.2), and data extraction (Section 2.3). Additionally, we used Parsifal [24] throughout the various stages to streamline the process.

2.1 Research questions

To systematically map the state of software repository classification research, we formulated seven research questions (RQs), described below. For each RQ we describe the dimension used to answer it, collectively defining our classification scheme (Table 1).

RQ1: Where and when was the research published. Understanding the publication context is essential for grasping the development and current state of software repository classification research. We identified two key dimensions:

Year of publication: Recording the publication year of each study allows observing temporal trends and assess whether interest in the topic is increasing, stable, or declining. When a paper has multiple versions (such as journal extensions), we only consider the latest peer-reviewed publication, to avoid overcounting the same result.

Venue: Identifying the specific conferences or journals where studies are published helps us understand the research fields and communities engaged with this topic. This dimension includes the name of the peer-reviewed venue (*e.g.*, *IEEE Transactions on Software Engineering*), providing insight into the fields that are contributing to this area (*e.g.*, software engineering, machine learning).

RQ2: What is the goal of the classification? This research question maps the motivations that drive researchers to engage in software repository classification and the practical applications of these classifications within the studies.

In particular, we identified two dimensions to answer this research question:

Stated motivation for classification: This dimension captures the authors' reasons for performing the classification, reflecting their desired outcomes or the problems they aim to address. For example, a study may state that the motivation is to enhance the discoverability of software repositories for educational purposes.

Application of the classification: This analysis focuses on the application of classification within the study, which may or may not correspond with the declared objectives. The classification could be used for various purposes, such as developing a recommender system, improving repositories with missing tags, or performing clustering for analytical insights. For example, a study could incorporate a tagging framework designed to optimize discoverability, thereby directly utilizing classification to achieve its stated goals.

By examining the *stated motivation* behind the research and the *applications of classification* methods in the literature, we aim to evaluate how well the implemented solutions align with the initial objectives.

RQ3: What kind of raw information is used for the classification? We explore this RQ with a single dimension:

Type of raw information used: data sources utilized, such as source code, README files, and commit histories. Understanding these inputs reveals which types of data are most commonly employed and how they contribute to the classification process.

Table 1: Classification scheme

Dimension	Scale
RQ1: Where and when was the research published?	
Year of publication	Time-based scale
Venue	Single-choice close scale
RQ2: What is the goal of the classification?	
Stated motivation for classification	Multi-valued open scale
Application of the classification	Multi-valued open scale
RQ3: What kind of raw information is used for the classification?	
Type of raw information used	Multi-valued open scale
RQ4: What features are computed for classification?	
Input features	Multi-valued open scale
RQ5: What are the output classes for the classification?	
Type of labels	Multi-valued closed scale
Class structure	Single-choice closed scale
Classification domain	Multi-valued open scale
RQ6: What is the training process?	
Data provenance	Multi-valued open scale
Use of preexisting datasets	Yes/No scale
Size of the training datasets	Integer scale
Types of supervision	Multi-valued closed scale
Label acquisition method	Multi-valued closed scale
Machine learning algorithms	Multi-valued open scale
RQ7: What is the evaluation process?	
Data provenance	Multi-valued open scale
Use of preexisting datasets	Yes/No scale
Size of the evaluation datasets	Integer scale
Label acquisition method	Multi-valued closed scale
Evaluation procedure	Multi-valued open scale
Evaluation metrics	Multi-valued open scale

RQ4: What features are computed for the classification? This research question investigates the features derived from raw data for the classification tasks. Analyzing these transformations helps map prevalent feature engineering practices and types of features commonly utilized in the field.

Input features: This dimension focuses on the techniques employed to represent raw information in a format suitable for machine learning models. Examples include embeddings, term frequency-inverse document frequency (TF-IDF) matrices, and statistical measures. Understanding these transformations provides insights into common approaches to feature engineering and their role in classification tasks.

RQ5: What are the output classes for the classification? This research question investigates the output classes used in classification tasks. This involves analyzing the types of labels, the nature of the classes, and the specific domains of classification. By mapping the output classes, we aim to understand how classification results are structured and identify trends within the target sets for different classification endeavors. To analyze the nature of classification outputs, we define three dimensions:

Type of labels: This dimension characterizes the classification task based on the structure of labels, including single-label versus multi-label and binary versus multi-class classifications.

Class structure (open vs closed classes): This dimension specifies whether the classification operates within a predefined set of categories (*closed-class*) or allows for the creation of new categories as needed (*open-class*). For example, classifying repositories as malware or not is a single-label, binary classification; assigning relevant concepts to a repository is a multi-label, open-class classification.

Classification domain: This dimension specifies the target categories or domains.

RQ6: What is the training process? This research question examines the training processes used to develop classification models, focusing on fundamental dimensions such as *data provenance*, *use of existing datasets*, *training set size*, *types of supervision*, *methods of label acquisition*, and *machine learning algorithms*. By analyzing these aspects, we aim to understand the common methodological practices and identify trends in training classification models.

Data provenance: This dimension highlights the sources of training data, such as GitHub, GitLab, or SourceForge, which offers insights into the diversity and representativeness of datasets. For instance, datasets obtained from GitHub often encompass a wide range of open source projects.

Use of preexisting datasets: whether pre-existing datasets are utilized, which is crucial for ensuring the reproducibility of previous results and enabling comparative evaluations.

Size of the training datasets: scale of training datasets, providing insights into trends related to model robustness and scalability. For example, training on thousands of repositories can lead to better generalization compared to using smaller datasets.

Types of supervision: used learning paradigm, distinguishing among supervised, unsupervised, semi-supervised, reinforcement, and self-supervised learning approaches.

Label acquisition method: how labels are obtained in supervised learning tasks—whether through manual annotation, automated processes, pre-existing datasets, or crowd-sourcing—thus helping to evaluate data quality and labeling practices.

Machine learning algorithms: techniques employed, ranging from traditional methods such as decision trees and support vector machines (SVM) to advanced approaches such as neural networks, graph neural networks (GNNs), transformers, and pre-trained language models.

RQ7: What is the evaluation process? Finally, this research question examines the evaluation processes employed to assess the performance of classification models. This includes analyzing the *evaluation dataset provenance*, the *use of preexisting datasets*, *dataset sizes*, *label acquisition methods*, *evaluation procedures*, and *evaluation metrics* dimensions. By mapping these dimensions, we aim to understand how performance is measured and reported in the field and identify trends in evaluation practices.

Data provenance: sources of evaluation data, such as GitHub, SourceForge, or proprietary datasets, providing insight into the generalizability of models across different contexts.

Use of preexisting datasets: whether preexisting datasets are used, which facilitates reproducibility and comparison across studies.

Size of the evaluation datasets: scale of datasets used for validation, as larger datasets tend to produce more robust and reliable performance metrics.

Label acquisition method: how labels are obtained for evaluation data—whether through manual annotation, automated processes, preexisting datasets, or crowd-sourcing—which can influence the interpretation of results.

Evaluation procedure: This dimension captures the validation methods employed, such as holdout validation, k-fold cross-validation, or bootstrapping. *Evaluation metrics*: This dimension highlights the criteria used to assess performance, including precision, recall, F1-score, accuracy, and other task-specific measures, reflecting the aspects of model performance emphasized in the studies.

2.2 Selection of primary sources

Figure 1 depicts the protocol used for the selection of our primary sources. First, we defined search queries to be used on several online libraries and executed them. After deduplication of obtained results, we applied inclusion and exclusion criteria, screened the titles and abstracts, screened the full content of the sources, and conducted a backward snowballing phase to obtain the final set of primary sources.

Search query definition. To identify the key terms for our search query, we adopted the PICO framework [12], commonly used in systematic reviews, which stands for "Population, Intervention, Comparison, and Outcome". This approach formulates research questions and structure as outlined by Petersen et al. [17]. Our systematic mapping study focused on the Population and Intervention components. The Population component in our study refers to software repositories and related entities. We defined this using the terms: *software*, *repository*, *library*, *project*, *application*, and *program*. The Intervention component pertains to the classification processes applied to these entities. We defined this using the terms: *classification*, *categorization*, *labeling*, *recommendation*, and *clustering*.

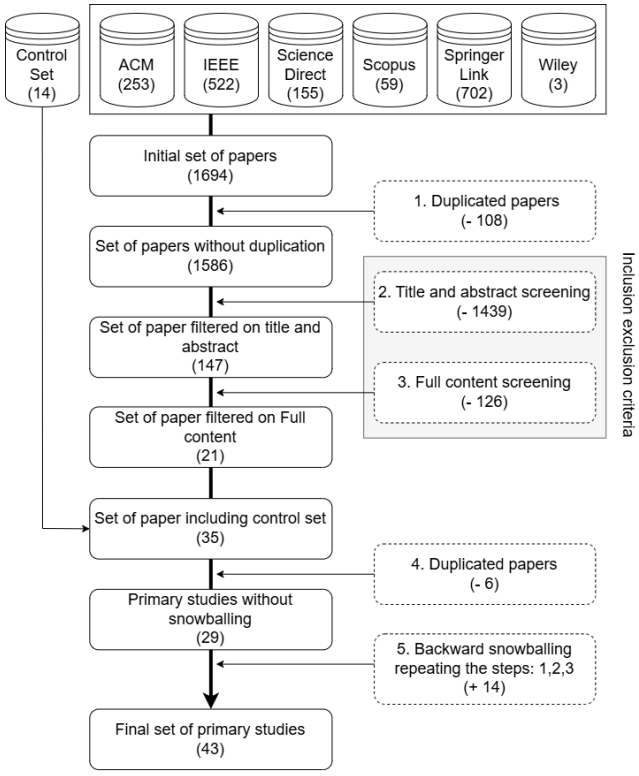

Figure 1: Protocol used for the selection of primary sources

We did not include the Comparision component because our study does not involve comparing different classification methods or interventions against each other. Our objective is to map the existing research on software repository classification as a whole rather than to evaluate or contrast specific approaches. Similarly, we excluded the Outcome component because we are not focusing on specific, measurable outcomes resulting from classification interventions, such as performance improvements or accuracy metrics. Instead, our aim is to identify and categorize the characteristics of existing studies, such as their motivations, data sources, features used, and methodologies, without assessing their empirical results.

According to the methodology of Biolchini et al. [5], we conducted an evaluation of our search strategy utilizing a curated control set comprising 14 papers, selected by the authors based on domain-specific expertise and experience in the field. Initially, our search retrieved 1684 papers, of which 108 were duplicates. We iteratively refined the query to optimize both precision and recall with respect to the control set. 6 of the 14 control papers were retrieved, producing a recall of 42.86%. The precision, calculated as the number of control papers retrieved over the total number of non-duplicated papers (6 out of 1576), was 0.37%.

The search string was refined, using the control papers as a benchmark, until an acceptable level of precision and recall was achieved. According to Beyer and Wright [4], the recall of search strategies can range from 0% to 87%, and precision from 0% to 14.3%. Our results, with a recall of 42.86% and precision of 0.37%, fall within this expected range. However, as highlighted by Wohlin and Prikladnicki [25], using a single search strategy can lead to missing studies, confirming the need to combine database searches with techniques such as snowball sampling to ensure more comprehensive coverage.

Table 2 presents the final search query, adapted to the syntax and semantics of each bibliographic database. These queries were applied exclusively to paper titles. This is because many of the keywords we search for are very common in the software engineering literature; searching for them in paper abstracts (or even full texts) returned large amounts of non-relevant papers during our iterations on the search query. This approach allowed us to ensure high relevance of the retrieved papers without compromising the breadth of the search. The selected digital libraries represent the major repositories available in the field of software engineering.

Inclusion and exclusion criteria. To ensure the selection of relevant studies, we applied the inclusion and exclusion criteria shown in Table 3. They require that studies: (C1) include a contribution that involves the automatic classification of software projects, (C2) be written in English, (C3) be peer-reviewed, (C4) be accessible in full text, and (C5) in cases of multiple versions of the same study, the most recent and complete version would be considered. Papers were excluded if they did not meet these criteria.

Title and abstract screening. We then applied a screening based on the titles and abstracts of the retrieved papers, as recommended by [18]. This step was performed by the first author, who read titles and abstracts, and applied inclusion/exclusion criteria *based on them*. Out of the 1586 total papers retained thus far, 1439 papers were rejected, leaving 147 potentially-relevant papers for further analysis.

Full content screening. We then applied the inclusion/exclusion criteria to the *full content* of retained papers. The task was divided among all authors. To assess the consistency of the screening process within the group, we randomly selected 10% of the papers—rounded up to 15 studies in total—and assigned these to five independent reviewers. Each reviewer applied the criteria to the same set of studies, and we calculated the inter-rater agreement using Fleiss' kappa coefficient. While Petersen and Ali [16] used Cohen's kappa, which is suitable for two raters, we opted for Fleiss' kappa as it accommodates multiple raters. The results indicated a kappa value of 0.370, which falls within the "fair agreement" range, suggesting that the selectors were largely consistent, with some variations in judgment, leaving room for improvement.

After discussion, all reviewers reached a consensus on the final decisions for the 15 studies, with only one paper being accepted. This led to a refinement of our criteria to enhance consistency in future evaluations, emphasizing the importance of inter-rater reliability in systematic mapping studies, where subjectivity can influence results.

Following the kappa evaluation, we proceeded with the evaluation of the remaining $147 - 15 = 132$ papers. These were divided equally among the five evaluators, each tasked with applying the inclusion and exclusion criteria to their subset.

The selection process allowed reviewers to mark studies as "maybe" in case of uncertainty about their inclusion, prompting a second evaluation by another reviewer. Out of the 132 studies,

Table 2: Queries used to search bibliographic databases.

Database	Search query
Springer Link	"Github Recommendation" OR "Github Topics" OR "Recommendation Repositories" OR "Topic Recommendation" OR "Software Classification" OR "Tagging Repository" OR "Repository Classification"
ACM	(Github AND Recommendation OR Github AND Topics OR Recommendation AND Repositories OR Topic AND Recommendation OR Software AND Classification OR Tagging AND Repository OR Repository AND Classification)
IEEE	"Github Recommendation" OR "Github Topics" OR "Recommendation Repositories" OR "Topic Recommendation" OR "Software Classification" OR "Tagging Repository" OR "Repository Classification"
Wiley	Github AND Recommendation OR Github AND Topics OR Recommendation AND Repositories OR Topic AND Recommendation OR Software AND Classification OR Tagging AND Repository OR Repository AND Classification
Science Direct	(Github Recommendation OR Tagging Repository OR Recommendation Repositories OR Topic Recommendation OR Software Classification OR Tagging Repository)
Scopus	("Github Recommendation" OR "Github Topics" OR "Recommendation Repositories" OR "Topic Recommendation" OR "Software Classification" OR "Tagging Repository" OR "Repository Classification")

Table 3: Inclusion and exclusion criteria.

Inclusion	Exclusion
Contribution Content	
Articles that include in their contributions a step where a set of software repositories are being automatically classified.	Articles that do not include this feature in their contributions.
Language	
Articles presented in English.	Articles not presented in English.
Source Type	
Articles from peer-reviewed sources.	Articles from grey literature (e.g., technical reports, theses, books, abstracts, presentations, tutorials, guidelines, summaries of conferences/editorials). Articles not peer-reviewed.
Accessibility	
Articles available in full text.	Articles not available in full text.
Extension	
If multiple publications of the same study exist presenting the same analysis, the latest version (i.e., most complete study report) will be included.	Studies for which a newer/more complete version exists.

105 were clear rejections, and 11 clear acceptances. 16 studies were marked as "maybe". After second opinion by a different reviewer, 9 of the maybe-s were accepted and 7 rejected. In the end, we accepted a total of 20 papers from the original 132.

In the end, we retained 35 papers for further analysis: 1 accepted during the kappa evaluation, 20 from the main selection phase, and 14 from the control set; further reduced to 29 papers after deduplication (note that the previous step of deduplication did not include the control set).

Backward snowballing. Considering the initial precision and recall, there was still room for improvement in the retrieval of relevant studies. To address this, we incorporated an additional step: *backward snowballing*. This process was applied to the 29 studies accepted during the selection of primary sources.

The snowballing phase followed the established methodology used for evaluating the primary sources: we began with 29 direct studies, which led to the identification of 990 references. These were processed through the three sequential steps: (1) removal of *duplicate papers*, (2) *title and abstract screening*, and (3) *full content screening*. Through this process, we identified 14 *additional* studies, bringing the total number of primary sources to 43.

2.3 Data extraction

In the data extraction phase, we first randomly selected five papers from the 43 primary sources. Five evaluators independently read the full text of the papers and extracted the information corresponding to the dimensions and scales of Table 1. Then, all evaluators met to discuss their results and ensure that their interpretations of the different dimensions and scales were aligned. Based on this agreement, the first author proceeded to read and extract the information from the remaining papers.

Once the information was extracted from all papers, the five evaluators met again to homogenize the vocabulary used for the seven open scales following an open card sorting process [27]. For each open scale, the evaluators listed and consolidated the information extracted from the papers. For instance, for the scale "Application of the classification", we consolidated the extracted terms "*tagging repositories*", "*assign topics to repositories*", and "*tag repositories*" under the common term "*Assign topics to repositories*". Once the exhaustive list of values for each scale was established, the first author applied them to all papers.

For certain open scales, we further analyzed recurring patterns in the data, grouping them into distinct themes to provide additional depth and structure to the analysis. For instance, for the scale "Type of raw information used", we grouped all values related to repository metadata (*e.g.*, repository name, description, license)

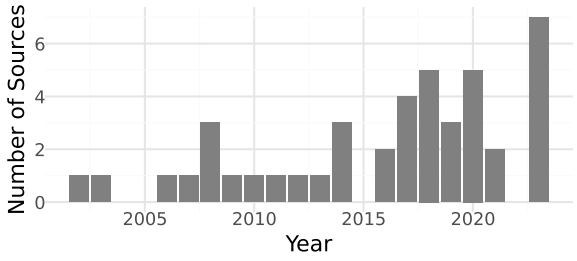

Figure 2: Publication years

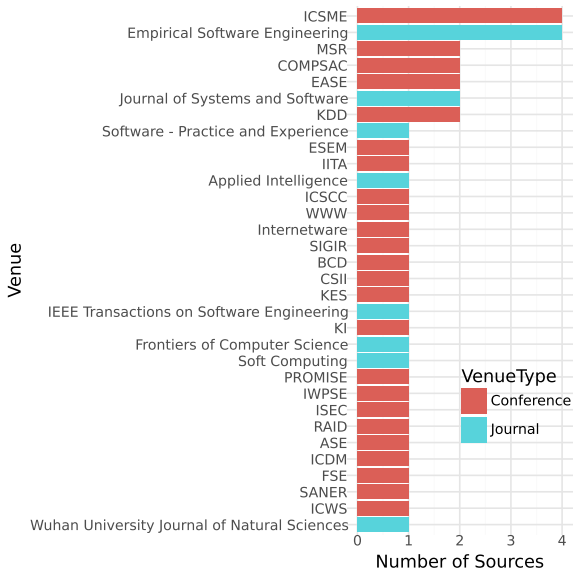

Figure 3: Publication venues

under the "Repository metadata" category. These themes and patterns are further discussed in Section 3, providing a comprehensive interpretation of the findings.

3 Results

In this section, we review and analyze the information extracted from the 43 primary sources following the scales and dimensions identified for each research question in Table 1. For each dimension, the number of studies matching it is indicated in parentheses. The raw data, analysis scripts, and plots discussed in this section are available from the replication package [1].

3.1 RQ1: Where and when was the research published?

Year of publication. As shown in Figure 2, the analyzed primary sources span more than two decades, from 2002 to 2023. Over this period, the annual number of publications increases over time, with notable peaks in 2018, 2020, and 2023. These observations suggest a growing and sustained research interest in software repository classification: the field remains active and evolving.

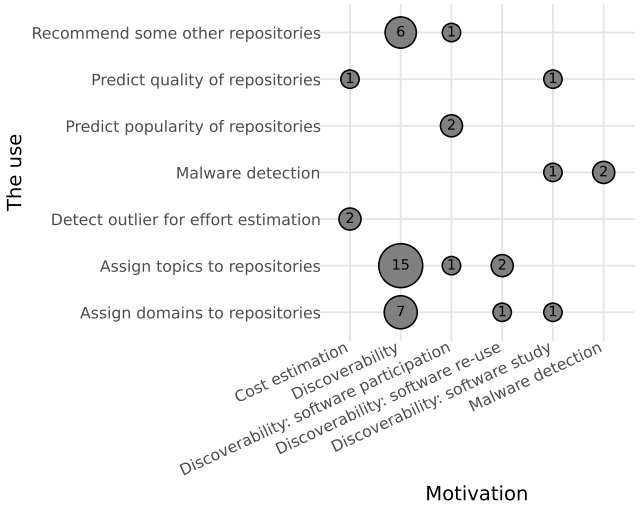

Figure 4: Motivations for and use of automatic classification in primary sources

Venue. Publication venues are quite varied, reflecting the interdisciplinary nature of research on software repository classification and the shared interests of various communities (Figure 3). While many papers are distributed across a wide range of venues, *Empirical Software Engineering* (4) and the *International Conference on Software Maintenance and Evolution* (4) published the most works, indicating a stronger involvement from the empirical software engineering community compared to others.

However, most venues appear only once or twice in the dataset, showcasing the broad distribution of studies throughout the publication landscape, including venues related to data mining (*e.g.*, *KDD, ICDM*), web services (*e.g.*, *ICWS*), and security (*e.g.*, *RAID*). This suggests that software repository classification is relevant to various computer science domains, rather than being concentrated in a single community or venue.

3.2 RQ2: What is the goal of the classification?

Stated motivation for classification. Stated motivations for classifying software repositories reveal a strong emphasis on enhancing the *discoverability* of software repositories, as depicted in Figure 4. The vast majority of studies mention *discoverability* (40) as a primary motivation for their work. For instance, some papers aim to improve repository navigation [S37], recommend relevant repositories [S65], or assign descriptive tags to enhance searchability [S55]. Discoverability is also sometimes linked to *education* (1) [S51], *software maintenance* (2) [S43], *software participation* (4) [S46], *software reuse* (3) [S31], and *software studies* (3) [S53]. Other motivations include *detection of malware* (2) and *effort estimation* (1).

Application of the classification. Looking at how classification is used in primary sources, a recurring theme is the use of classification to *assign topics to repositories* (18) to enhance the discoverability of repositories, thereby helping users navigate large software ecosystems [S32, S43, S55]. Here, topics are open-ended labels such

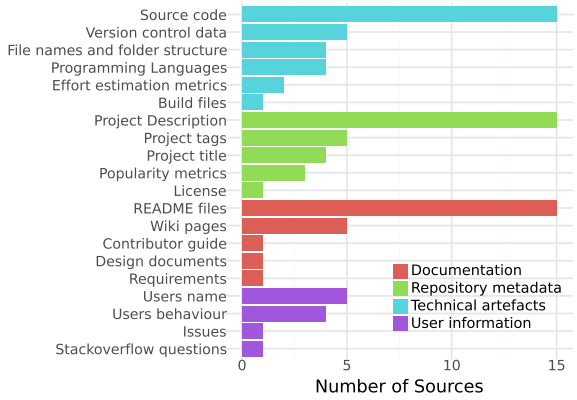

Figure 5: Input information used for the classification. Some approaches combine information from different sources.

as GitHub Topics.[2] Additionally, studies *assign domains to repositories* (9); domains are smaller close-ended sets of labels representing software categories or application domains. Another notable area of focus is the *recommendation of repositories* (7), based on either content similarity [S28] or developers' behavioral patterns [S38].

3.3 RQ3: What kind of raw information is used for the classification?

As depicted in Figure 5, the primary sources analyzed rely on a variety of raw information from software repositories to classify them, from textual documentation to technical artifacts through repository metadata and user social information.

The most commonly used raw data is *source code* (15), together with *project descriptions* (15) and *README files* (15). Other frequently used sources include *version control data* (7) such as commits. It is noteworthy that raw inputs are often used in combination. For instance, studies frequently leverage both *source code* and *README files* [S65], or they combine textual data with *version control information* to enrich classification input [S34]. This multi-sources approach captures diverse perspectives on repositories to improve the accuracy of the classification.

In Figure 5 we grouped raw information into four macro categories—*documentation*, *repository metadata*, *technical artifacts*, and *user information*—providing a structured view of how different features contribute to repository classification.

3.4 RQ4: What features are computed for classification?

Our analysis (see Figure 6) reveals that the most commonly computed features for classification are *co-occurrence matrices* (18). These matrices capture relationships between elements, such as terms in textual data, and are often employed as inputs for methods like term frequency-inverse document frequency (TF-IDF) or latent Dirichlet allocation (LDA).

[2]https://github.com/topics

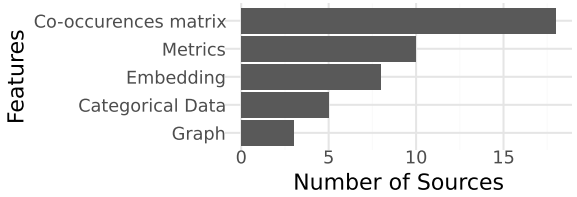

Figure 6: Features used for the classification

Metrics (10) emerge as the second most common feature type. This category encompasses various numerical and statistical properties derived from raw inputs, including code measures, repository size or user activity levels. Closely trailing metrics are *embedding techniques* (8). These methods transform raw data, such as textual or structural inputs, into dense vector representations. Embeddings are particularly valued for their ability to capture semantic relationships, making them especially useful for analyzing textual artifacts like *README files* or project descriptions.

In addition to these commonly used features, *categorical data* (5) and *graph-based features* (3) appear less frequently. Categorical data consists of discrete labels or categories assigned based on specific attributes, whereas graph-based features represent relationships and model structural dependencies or interactions between entities within the data.

3.5 RQ5: What are the output classes for the classification?

Type of labels. Our analysis reveals that most studies adopt a *multilabel* (27) classification approach, allowing repositories to belong to multiple categories simultaneously. This approach is particularly effective for complex tasks, such as topic assignment or repository recommendation, where repositories may span several themes or purposes [S50, S62]. In contrast, *single-label classifications* (16) assign each repository to exactly one category. This simpler approach is often applied to binary classifications or scenarios where categories are mutually exclusive [S35].

Class structure (open vs closed classes). Regarding the structure of the classes, *closed-class* (30) classifications dominate. In this case, repositories are classified into a predefined set of categories, such as known software domains [S43]. Conversely, *open-class* (13) classifications, which allow for the creation of new categories as needed, are particularly useful for recommender systems where the number of suggestions is not pre-defined [S65].

Classification Domain. The classification output domain provides insight into the intended purpose of the classification tasks. Our analysis reveals that *topics* (17) and *software categories* (12) are the most frequently assigned labels, indicating a strong focus on thematic organization and categorization of repositories [S67]. Less commonly, classifications target *repositories* (6), often for recommendation purposes [S39], or *quality metrics* (4), where the goal is to assess various aspects of software quality [S52].

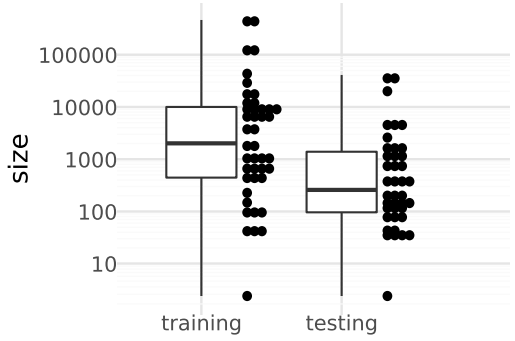

Figure 7: Sizes of the training and evaluation datasets

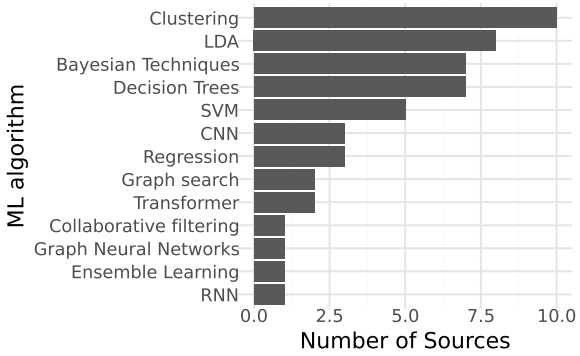

Figure 8: Algorithms used for the classification

3.6 RQ6: What is the training process?

Data provenance. The data used for training classification models predominantly originates from publicly available repositories, with *GitHub* (22) being the overwhelmingly dominant source. Other sources, such as *SourceForge* (11) and *private datasets* (3) from company-specific repositories, are far less common. Although a few studies use data from multiple sources, such as combining GitHub and StackOverflow, the overall diversity of data provenance remains limited, with most training sets heavily reliant on GitHub [S30, S34].

Use of preexisting datasets. Only a few primary sources use *preexisting datasets* (11), whereas the majority rely on datasets specifically created for their experiments. This dependence on custom datasets reflects a lack of standardized benchmarks for repository classification, which can interfere with reproducibility and comparability across studies.

Size of the training datasets. Training set sizes vary significantly, ranging from as few as 2 repositories [S48] to over 460 000 repositories in large-scale studies [S30], as shown in Figure 7 (left). Large datasets, such as those exceeding 100 000 repositories, are typically used in studies focused on scalable machine learning approaches (*e.g.*, [S34]), whereas smaller datasets are often seen in exploratory tasks (*e.g.*, [S68]).

Types of supervision. The most prevalent learning paradigm is *supervised learning* (29). *Unsupervised learning* (13) accounts for a smaller portion, primarily in clustering and topic modeling tasks. *weak supervision* (1) also appears, highlighting innovative strategies to reduce the dependency on labeled dataset [S69].

Label acquisition method. For supervised studies, labels are most commonly derived from *preexisting annotations* (27), as seen in studies leveraging datasets like GitHub Topics. Other methods include *computed labels* (11) (*e.g.*, inferred through algorithms) and *human labeled* (2). Preexisting and computed labels dominate, reflecting the efficiency and scalability of automated and semi-automated approaches.

Machine learning algorithms. Our analysis highlights a diverse range of machine learning algorithms employed in the studies (Figure 8). *Clustering* (10) emerges as the most frequently used technique, particularly for unsupervised tasks such as repository grouping and topic modeling. Its versatility makes it well-suited for exploratory data analysis, where predefined categories may not exist.

LDA (Latent Dirichlet Allocation) (8) is another widely used approach, especially when handling textual data like *README files* or project descriptions. Similarly, *Bayesian Techniques* (7) and *Decision Trees* (7) are commonly employed, often in combination with other methods, for instance, *SVMs (Support Vector Machines)* (5). *Regression* (3) is applied to prediction tasks, such as estimating repository popularity or assessing software quality [S29, S70]. Meanwhile, *CNNs (Convolutional Neural Networks)* (3) are used for their ability to extract features from both structured and unstructured data, making them particularly valuable in tasks involving complex inputs [S69]. Other significant methods include *Graph Search* (2), often applied to analyze relationships within data, and *Collaborative Filtering* (2), commonly used for recommendation tasks. Modern approaches like *Transformers* (2) are also beginning to appear in the field, showcasing their potential for advancing classification methodologies.

3.7 RQ7: What is the evaluation process?

Data provenance. Most studies utilize data from *GitHub* (22), highlighting its dominant role as the primary source of data for repository classification research. GitHub's extensive repository ecosystem, metadata, and activity logs make it an invaluable resource for a wide range of classification tasks. *SourceForge* (10) is the second most common source. Although less prevalent than GitHub, SourceForge contributed significantly to this research area.

A smaller number of studies use data from *private companies* (3), reflecting the use of proprietary datasets for specialized or domain-specific investigations. Similarly, *Ohloh* (2) is cited, emphasizing its role in aggregating open source project information. As for the training data provenance, less common sources include *Debian Packages* (1), *StackOverflow* (1), *F-Droid* (1), and *tera-PROMISE* (1).

Use of preexisting datasets. As with the training process, most studies *construct their own datasets* (30) for the evaluation process, which does not facilitate comparison between different methodologies. However, few studies utilize *preexisting datasets* (12), addressing this limitation to some extent.

Size of the evaluation datasets. The sizes of evaluation datasets vary significantly (Figure 7, right), ranging from as small as 2 [S48]

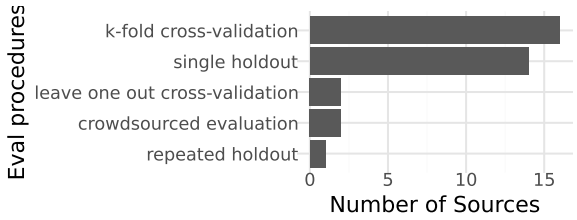

Figure 9: Evaluation procedures

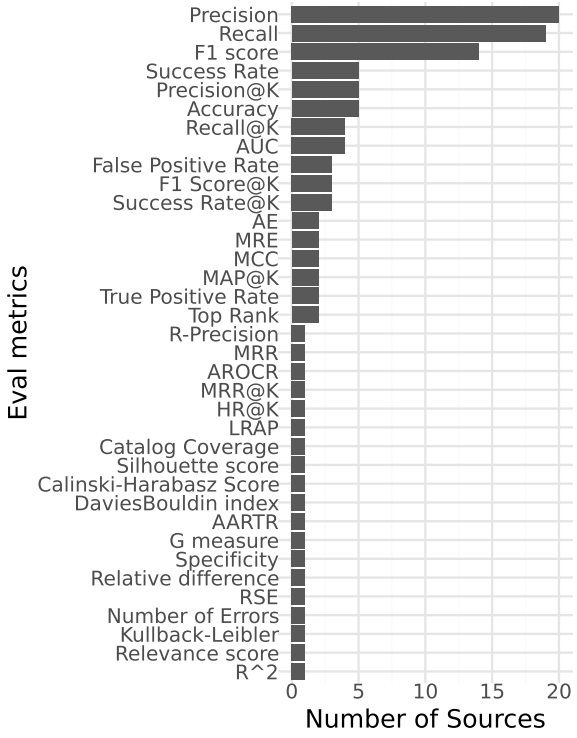

Figure 10: Evaluation metrics

repositories to over 40 000 in large-scale studies [S34], highlighting the need for further evaluation of real-world archive sizes and their impact on classification performance.

Label acquisition method. Labels come predominantly from *pre-existing annotations* (27), often derived from established sources like GitHub Topics. *Computed labels* (7) and *human-labeled datasets* (5) also play a significant role.

Evaluation procedure. K-fold cross-validation (16) is the most widely used evaluation method, commonly employing values of k such as 10 and 5 (see Figure 9). *Single holdout* (14) is also frequent, with variations in specific ratios such as *80/20* or *90/10*. Other methods, including *leave one out cross-validation* (2), *crowdsourced evaluation* (2), and *repeated holdout* (1) are less commonly used and focus on tailored approaches suited for specific study designs.

Evaluation metrics. The evaluation metrics used highlight the diverse goals and methodologies in software repository classification (see Figure 10). *Precision* (20) is the most commonly used metric, often in conjunction with *Recall* (19). *F1 Score* (14) is particularly significant in situations where achieving a balance between these two metrics is essential. Other metrics for a fixed number of classes include *Accuracy* (5) and *Success Rate* (5). Additionally, ranking-specific measures such as *Recall@K* (4) and *Precision@K* (5) highlight the emphasis on recommendation systems and ranking tasks. However, most metrics only appear once in the studied sources, highlighting the lack of a unified methodology for evaluating classification tasks with negative effects on the ability to compare results.

4 Discussion

Our study reveals trends and gaps in the automatic classification of software repositories, as detailed below.

A growing research area. Analyzing publication years, we observe a positive growth trajectory, with increasing interest in this topic, particularly within the software engineering community. Overall, 60% of the papers were published in the last seven years of the study period. This rising interest aligns with the rapid expansion of software hosted on development platforms like GitHub and archives like Software Heritage. Notably, this growth underscores the significant challenge of discoverability—the most frequently cited motivation for studies in this field.

Discoverability in practice. While discoverability is a primary driver of repository classification, care must be taken to ensure the solutions proposed are aligned to actual developer needs. At the moment, discoverability is often formulated as a classification task whose format is strongly influenced by available data: either as a classification of broad software *domains* (mostly obtained from SourceForge or manually curated), or as a classification of fine-grained *topics* (mostly obtained from larger datasets extracted from SourceForge or Github). Since 2019, only works on GitHub topics (akin to tags) have been published. Studies of general tagging systems have found that in search scenarios, not all users found tagging systems useful. Notably, some users felt that tags are either too broad or too narrow for their search needs [11]. Other work found that tag clouds were most suited to searches for specific information, rather than general information [21], echoing a similar sentiment. Of note, a third type of approaches focuses on recommending similar repositories to an existing one: this scenario differs markedly from the previous two. In light of the above, we call on the community to ensure that discoverability tasks are well aligned with practical goals, by means of user studies.

Another use of discoverability is in the context of mining software repository studies, where researchers seek to select a relevant and representative sample of repositories to analyze, and then extrapolate more general conclusions from. We found surprisingly few studies that addressed this need, despite the fact that understanding application domains and categories is crucial for qualitative and quantitative analyses of the extracted data. This finding echoes a literature review on the workflows and methodologies in mining studies [23], which finds that MSR studies tend to use simple sampling criteria such as programming language or popularity.

Beyond discoverability. We were very surprised that discoverability would be such a dominating topic: only 12% of papers address

other topics. In addition to cost estimation and malware detection, there are likely other scenarios that can benefit from increased research (e.g. librairies or packages at risk of being abandoned [2]).

Data and data quality. Regarding the data used for training and evaluation, most studies operate on dataset of limited sizes, raising questions about the scalability of these methodologies when applied to real-world, large-scale repository collections. For instance, the largest dataset identified in this study contains approximately 460 000 repositories, 3 orders of magnitudes away from the 300 million repositories archived by Software Heritage. Whereas it is understandable that researchers work on smaller datasets, it would be useful to *also* have studies that convincingly show the practical applicability of automated classifiers to larger, real-world scenarios.

Regarding GitHub topics, while there is a large quantity of data available, making them appealing, there are some known data quality issues (e.g., some topics specify the programming language the repository is implemented in—a task better fulfilled by programming language identification tools than repository classification) [S55]. These issues contribute to the possible misalignment between the task as formulated and the practical uses by developers.

Machine learning techniques. Furthermore, most studies utilize conventional machine learning methods such as decision trees and clustering. At the same time, only a handful of research studies have investigated more sophisticated approaches such as neural networks or transformers, which have recently shown considerable promise in other fields. Foundation models such as BERT [6] or CodeBERT [8] in particular have found success in software engineering tasks [13], even with small-scale datasets [19]. The advent of truly large language models [22], who in some cases can dispense with additional training opens up additional opportunities. So far, a single primary source is using pretrained models [S37], while we are aware of an as yet unpublished work (thus not fulfilling our inclusion critera) using an LLM to detect scientific software [15]. This limited exploration of recent approaches suggests a potential avenue for future research, in which scalability issues due to larger and more diverse datasets are more likely to emerge.

Lack of standard benchmarks. Another significant limitation is the absence of standardized benchmarks, including datasets, classification tasks, and evaluation metrics. This absence makes it difficult to compare the performance of various machine learning approaches for this task. Establishing standardized benchmarks would enhance reproducibility and provide a solid foundation for evaluating and improving the automatic classification of software repositories, as has been seen for other software engineering fields such as with benchmarks such as Defects4J [10] or the Bellon benchmark [3]. Furthermore, benchmarks are know to foster communities accelerate progress [20]. There are however pitfalls as benchmarks must be carefully constructed to minimize data quality issues and incentivize the right objectives. As such, given the alignment and data quality issues mentioned above, simply collecting a large amount of GitHub topics might not be sufficient to yield an effective benchmark. However, approaches that involve a degree of curation, such as GitRanking [S55], constitute a promising way forward.

5 Threats to validity

Like all secondary studies, this systematic mapping study is subject to validity concerns. We report these threats following the guidelines of Wohlin et al. [26].

Internal validity. To mitigate the potential threats to validity during the screening phase, we performed a pilot screening of *10%* of the papers with overlapping reviewers, followed by a discussion round to resolve discrepancies. We also computed the Fleiss kappa to measure inter-rater agreement among the five reviewers. During the data extraction phase, we avoided inconsistency in gathering data by having multiple rounds of discussion on how to interpret and apply dimensions and scales. A potential threat arises from the diversity of studies, particularly in *Machine Learning Algorithm* and *Evaluation metrics*. Similar metrics reported under different names were grouped into categories to streamline analysis, which may obscure subtle differences. To mitigate this, we employed an open card-sorting process for consistency.

External validity. Because our study focuses on peer-reviewed publications explicitly mentioning the automatic classification of software repositories, some relevant industrial or unpublished work (*e.g.*, gray literature) may have been missed. Furthermore, despite leveraging multiple digital libraries and following a rigorous protocol to craft search queries, we might still have missed relevant studies. The inclusion of backward snowballing mitigates this to some extent, but complete coverage is never guaranteed in any systematic review of the literature.

6 Conclusion

Our systematic mapping of 43 primary studies, filtered from an initial set of almost 1700 works, shows that the interest in software repository classification has grown steadily over the last two decades, with a marked increase in the past seven years. Although discoverability is the dominant motivation, relatively few studies address other objectives, such as security or maintainability classifications. We have seen a wide range of different sources, such as datasets, input data, and features, that highlight the complexity of large software repositories. In addition, there is significant variation in machine learning algorithms, evaluation methods, and metrics, which points to a lack of standardization in this field. Establishing common benchmarks would help in improving the comparability of research results, which is currently quite difficult. Among the gaps we have observed in the literature, three stand out: (1) the need of closing the gap between stated goals for repository classification and developer needs; (2) empirical verifications of the scalability of state-of-the-art approaches to the largest existing scale of repository collection; (3) the application of state-of-the-art machine learning techniques (e.g., transformers) to this task.

References

[1] Anonymous. 2025. Automatic Classification of Software Repositories: A Systematic Mapping Study - replication package. https://doi.org/10.5281/zenodo.14773537

[2] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, Porto de Galinhas, Brazil, 1–12. https://doi.org/10.1109/ESEM.2019.8870181 ISSN: 1949-3789.

[3] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. 2007. Comparison and Evaluation of Clone Detection Tools. *IEEE Transactions on Software Engineering* 33, 9 (Sept. 2007), 577–591. https://doi.org/10.1109/TSE.2007.70725 Conference Name: IEEE Transactions on Software Engineering.

[4] Fiona Beyer and Kath Wright. 2011. Comprehensive searching for systematic reviews: a comparison of database performance. *York: Centre for Reviews and Dissemination, University of York* (2011).

[5] Jorge Calmon de Almeida Biolchini, Paula Gomes Mian, Ana Cândida Cruz Natali, Tayana Uchôa Conte, and Guilherme Horta Travassos. 2007. Scientific research ontology to support systematic review in software engineering. *Adv. Eng. Informatics* 21, 2 (2007), 133–151. https://doi.org/10.1016/J.AEI.2006.11.006

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[7] Roberto Di Cosmo and Stefano Zacchiroli. 2017. Software Heritage: Why and How to Preserve Software Source Code. In *iPRES 2017 - 14th International Conference on Digital Preservation*. PHAIDRA, Kyoto, Japan, 1–10. https://hal.science/hal-01590958

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[9] Joseph L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin* 76, 5 (1971), 378–382. https://doi.org/10.1037/h0031619 Place: US Publisher: American Psychological Association.

[10] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: a database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014)*. Association for Computing Machinery, New York, NY, USA, 437–440. https://doi.org/10.1145/2610384.2628055

[11] Margaret Kipp and D. Grant Campbell. 2010. Searching with Tags: Do Tags Help Users Find Things? *Knowledge Organization* 37 (Jan. 2010), 239–255. https://doi.org/10.5771/0943-7444-2010-4-239

[12] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering – A systematic literature review. *Information and Software Technology* 51, 1 (Jan. 2009), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

[13] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models. In *Proceedings of 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, Madrid, ES, 324–335. https://doi.org/10.1109/ICSE43902.2021.00040 ISSN: 1558-1225.

[14] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. 2019. World of Code: An Infrastructure for Mining the Universe of Open Source VCS Data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, Montreal, Canada, 143–154. https://doi.org/10.1109/MSR.2019.00031 ISSN: 2574-3864.

[15] Addi Malviya-Thakur, Reed Milewicz, Lavinia Paganini, Ahmed Samir Imam Mahmoud, and Audris Mockus. 2023. SciCat: A Curated Dataset of Scientific Software Repositories. https://doi.org/10.48550/arXiv.2312.06382 arXiv:2312.06382 [cs].

[16] Kai Petersen and Nauman Bin Ali. 2011. Identifying Strategies for Study Selection in Systematic Reviews and Maps. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM 2011, Banff, AB, Canada, September 22-23, 2011*. IEEE, Banff, Canada, 351–354. https://doi.org/10.1109/ESEM.2011.46

[17] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*. Association for Computing Machinery, Bari, Italy, 68–77. event-place: Italy.

[18] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (Aug. 2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

[19] Julian Aron Prenner and Romain Robbes. 2022. Making the Most of Small Software Engineering Datasets With Modern Machine Learning. *IEEE Transactions on Software Engineering* 48, 12 (Dec. 2022), 5050–5067. https://doi.org/10.1109/TSE.2021.3135465 Conference Name: IEEE Transactions on Software Engineering.

[20] S.E. Sim, S. Easterbrook, and R.C. Holt. 2003. Using benchmarking to advance research: a challenge to software engineering. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, Portland, USA, 74–83. https://doi.org/10.1109/ICSE.2003.1201189 ISSN: 0270-5257.

[21] James Sinclair and Michael Cardew-Hall. 2008. The folksonomy tag cloud: when is it useful? *Journal of Information Science* 34, 1 (Feb. 2008), 15–29. https://doi.org/10.1177/0165551506078083 Publisher: SAGE Publications Ltd.

[22] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. https://doi.org/10.48550/arXiv.2302.13971 arXiv:2302.13971 [cs].

[23] Adam Tutko, Austin Z. Henley, and Audris Mockus. 2022. How are Software Repositories Mined? A Systematic Literature Review of Workflows, Methodologies, Reproducibility, and Tools. https://doi.org/10.48550/arXiv.2204.08108 arXiv:2204.08108 [cs].

[24] Will Segatto Vitor Freitas. 2024. Parsifal. https://parsif.al/

[25] Claes Wohlin and Rafael Prikladnicki. 2013. Systematic literature reviews in software engineering. *Inf. Softw. Technol.* 55, 6 (2013), 919–920.

[26] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering* (2012 ed.). Springer, Berlin, Germany.

[27] T. Zimmermann. 2016. Card-sorting: From text to themes. In *Perspectives on Data Science for Software Engineering*, Tim Menzies, Laurie Williams, and Thomas Zimmermann (Eds.). Morgan Kaufmann, Boston, 137–141. https://doi.org/10.1016/B978-0-12-804206-9.00027-1

Primary sources

[S28] Doaa Altarawy, Hossameldin Shahin, Ayat Mohammed, and Na Meng. 2018. Lascad : Language-agnostic software categorization and similar application detection. *Journal of Systems and Software* 142 (Aug. 2018), 21–34. https://doi.org/10.1016/j.jss.2018.04.018

[S29] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Predicting the Popularity of GitHub Repositories. In *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2016)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/2972958.2972966

[S30] Xuyang Cai, Jiangang Zhu, Beijun Shen, and Yuting Chen. 2016. GRETA: Graph-Based Tag Assignment for GitHub Repositories. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, Atlanta, USA, 63–72. https://doi.org/10.1109/COMPSAC.2016.124 ISSN: 0730-3157.

[S31] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong Nguyen, and Riccardo Rubei. 2020. TopFilter: An Approach to Recommend Relevant GitHub Topics. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Bari Italy, 1–11. https://doi.org/10.1145/3382494.3410690

[S32] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T. Nguyen, and Riccardo Rubei. 2023. HybridRec: A recommender system for tagging GitHub repositories. *Applied Intelligence* 53, 8 (April 2023), 9708–9730. https://doi.org/10.1007/s10489-022-03864-y

[S33] Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Phuong T. Nguyen. 2020. A Multinomial Naïve Bayesian (MNB) Network to Automatically Recommend Topics for GitHub Repositories. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (EASE '20)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/3383219.3383227

[S34] Junxiao Han, Shuiguang Deng, Xin Xia, Dongjing Wang, and Jianwei Yin. 2019. Characterization and Prediction of Popular Projects on GitHub. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, Milwaukee, WI, USA, 21–26. https://doi.org/10.1109/COMPSAC.2019.00013 ISSN: 0730-3157.

[S35] Kazunori Iwata, Toyoshiro Nakashima, Yoshiyuki Anan, and Naohiro Ishii. 2019. Applying Machine Learning Classification to Determining Outliers in Effort for Embedded Software Development Projects. In *2019 6th International Conference on Computational Science/Intelligence and Applied Informatics (CSII)*. IEEE, Honolulu, USA, 78–83. https://doi.org/10.1109/CSII.2019.00021

[S36] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. 2021. Topic recommendation for software repositories using multi-label classification algorithms.

Empirical Software Engineering 26, 5 (July 2021), 93. https://doi.org/10.1007/s10664-021-09976-2

[S37] Maliheh Izadi, Mahtab Nejati, and Abbas Heydarnoori. 2023. Semantically-enhanced topic recommendation systems for software projects. *Empirical Software Engineering* 28, 2 (Feb. 2023), 50. https://doi.org/10.1007/s10664-022-10272-w

[S38] Jyun-Yu Jiang, Pu-Jen Cheng, and Wei Wang. 2017. Open Source Repository Recommendation in Social Coding. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Shinjuku Tokyo Japan, 1173–1176. https://doi.org/10.1145/3077136.3080753

[S39] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue. 2003. Automatic categorization algorithm for evolvable software archive. In *Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings.* IEEE, Helsinki, Finland, 195–200. https://doi.org/10.1109/IWPSE.2003.1231227

[S40] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue. 2004. MUDABlue: an automatic categorization system for open source repositories. In *11th Asia-Pacific Software Engineering Conference*. IEEE, Busan, Korea, 184–193. https://doi.org/10.1109/APSEC.2004.69 ISSN: 1530-1362.

[S41] Yesol Kim, Seong-je Cho, Sangchul Han, and Ilsun You. 2018. A software classification scheme using binary-level characteristics for efficient software filtering. *Soft Computing* 22, 2 (Jan. 2018), 595–606. https://doi.org/10.1007/s00500-016-2357-x

[S42] Naoki Kinoshita, Akito Monden, Masateru Tshunoda, and Zeynep Yücel. 2018. Predictability Classification for Software Effort Estimation. In *2018 IEEE International Conference on Big Data, Cloud Computing, Data Science & Engineering (BCD)*. IEEE, Yonago, Japan, 43–48. https://doi.org/10.1109/BCD2018.2018.00015

[S43] Alexander LeClair, Zachary Eberhart, and Collin McMillan. 2018. Adapting Neural Text Classification for Improved Software Categorization. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Madrid, Spain, 461–472. https://doi.org/10.1109/ICSME.2018.00056 ISSN: 2576-3148.

[S44] Jun-Tao Liang and Xiao-Yuan Jiang. 2008. A Software Component Classification Based on Facet and Neural Network. In *2008 International Symposium on Intelligent Information Technology Application Workshops*. IEEE, Shanghai, China, 1121–1123. https://doi.org/10.1109/IITA.Workshops.2008.277

[S45] Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. 2014. On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering* 19, 3 (June 2014), 582–618. https://doi.org/10.1007/s10664-012-9230-z

[S46] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. 2007. Mining concepts from code with probabilistic topic models. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, Atlanta Georgia USA, 461–464. https://doi.org/10.1145/1321631.1321709

[S47] Yuzhan Ma, Sarah Fakhoury, Michael Christensen, Venera Arnaoudova, Waleed Zogaan, and Mehdi Mirakhorli. 2018. Automatic classification of software artifacts in open-source applications. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, Gothenburg Sweden, 414–425. https://doi.org/10.1145/3196398.3196446

[S48] Girish Maskeri, Santonu Sarkar, and Kenneth Heafield. 2008. Mining business topics in source code using latent dirichlet allocation. In *Proceedings of the 1st India software engineering conference*. ACM, Hyderabad India, 113–120. https://doi.org/10.1145/1342211.1342234

[S49] Collin McMillan, Mario Linares-Vásquez, Denys Poshyvanyk, and Mark Grechanik. 2011. Categorizing software applications for maintenance. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, Williamsburg, USA, 343–352. https://doi.org/10.1109/ICSM.2011.6080801 ISSN: 1063-6773.

[S50] N Nalini, S Rishabh, Bhargav D Bhat, Vaibhav Jamwal, and Aayush Kumar. 2023. Github Recommendation System And User Analytics. In *2023 9th International Conference on Smart Computing and Communications (ICSCC)*. IEEE, Kochi, India, 582–587. https://doi.org/10.1109/ICSCC59169.2023.10334939

[S51] Mathieu Nassif and Martin P. Robillard. 2023. Identifying Concepts in Software Projects. *IEEE Transactions on Software Engineering* 49, 7 (July 2023), 3660–3674. https://doi.org/10.1109/TSE.2023.3265855 Conference Name: IEEE Transactions on Software Engineering.

[S52] Peter Pickerill, Heiko Joshua Jungen, Mirosław Ochodek, Michał Maćkowiak, and Miroslaw Staron. 2020. PHANTOM: Curating GitHub for engineered software projects using time-series clustering. *Empirical Software Engineering* 25, 4 (July 2020), 2897–2929. https://doi.org/10.1007/s10664-020-09825-8

[S53] Md Omar Faruk Rokon, Risul Islam, Ahmad Darki, Evangelos E. Papalexakis, and Michalis Faloutsos. 2020. {SourceFinder}: Finding Malware {Source-Code} from Publicly Available Repositories in {GitHub}. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. USENIX Association, Donostia, Spain, 149–163. https://www.usenix.org/conference/raid2020/presentation/omar

[S54] Md Omar Faruk Rokon, Pei Yan, Risul Islam, and Michalis Faloutsos. 2021. Repo2Vec: A Comprehensive Embedding Approach for Determining Repository Similarity. In *Proceedings of 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Luxembourg, 355–365. https://doi.org/10.1109/ICSME52107.2021.00038

[S55] Cezar Sas, Andrea Capiluppi, Claudio Di Sipio, Juri Di Rocco, and Davide Di Ruscio. 2023. GitRanking: A ranking of GitHub topics for software classification using active sampling. *Software: Practice and Experience* 53, 10 (Oct. 2023), 1982–2006. https://doi.org/10.1002/spe.3238

[S56] Huajie Shao, Dachun Sun, Jiahao Wu, Zecheng Zhang, Aston Zhang, Shuochao Yao, Shengzhong Liu, Tianshi Wang, Chao Zhang, and Tarek Abdelzaher. 2020. paper2repo: GitHub Repository Recommendation for Academic Papers. In *Proceedings of The Web Conference 2020 (WWW '20)*. Association for Computing Machinery, New York, NY, USA, 629–639. https://doi.org/10.1145/3366423.3380145

[S57] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. 2017. Cataloging GitHub Repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE '17)*. Association for Computing Machinery, New York, NY, USA, 314–319. https://doi.org/10.1145/3084226.3084287

[S58] Marcus Soll and Malte Vosgerau. 2017. ClassifyHub: An Algorithm to Classify GitHub Repositories. In *KI 2017: Advances in Artificial Intelligence (Lecture Notes in Computer Science)*, Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm (Eds.). Springer International Publishing, Cham, 373–379. https://doi.org/10.1007/978-3-319-67190-1_34

[S59] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. 2009. Using Latent Dirichlet Allocation for automatic categorization of software. In *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, Vancouver, Canada, 163–166. https://doi.org/10.1109/MSR.2009.5069496 ISSN: 2160-1860.

[S60] Secil Ugurel, Robert Krovetz, and C. Lee Giles. 2002. What's the code?: automatic classification of source code archives. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, Edmonton Alberta Canada, 632–638. https://doi.org/10.1145/775047.775141

[S61] Tao Wang, Huaimin Wang, Gang Yin, Charles X. Ling, Xiang Li, and Peng Zou. 2013. Mining Software Profile across Multiple Repositories for Hierarchical Categorization. In *2013 IEEE International Conference on Software Maintenance*. IEEE, Eindhoven, Netherlands, 240–249. https://doi.org/10.1109/ICSM.2013.35 ISSN: 1063-6773.

[S62] Tao Wang, Huaimin Wang, Gang Yin, Charles X. Ling, Xiao Li, and Peng Zou. 2014. Tag recommendation for open source software. *Frontiers of Computer Science* 8, 1 (Feb. 2014), 69–82. https://doi.org/10.1007/s11704-013-2394-x

[S63] Tao Wang, Gang Yin, Xiang Li, and Huaimin Wang. 2012. Labeled topic detection of open source software from mining mass textual project profiles. In *Proceedings of the First International Workshop on Software Mining (SoftwareMining '12)*. Association for Computing Machinery, New York, NY, USA, 17–24. https://doi.org/10.1145/2384416.2384419

[S64] Ratnadira Widyasari, Zhipeng Zhao, Thanh Le Cong, Hong Jin Kang, and David Lo. 2023. Topic Recommendation for GitHub Repositories: How Far Can Extreme Multi-Label Learning Go?. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Taipa, Macao, 167–178. https://doi.org/10.1109/SANER56733.2023.00025 ISSN: 2640-7574.

[S65] Wenyuan Xu, Xiaobing Sun, Xin Xia, and Xiang Chen. 2017. Scalable Relevant Project Recommendation on GitHub. In *Proceedings of the 9th Asia-Pacific Symposium on Internetware (Internetware '17)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3131704.3131706

[S66] Yueshen Xu, Yuhong Jiang, Xinkui Zhao, Ying Li, and Rui Li. 2023. Personalized Repository Recommendation Service for Developers with Multi-modal Features Learning. In *2023 IEEE International Conference on Web Services (ICWS)*. IEEE, Chicago, USA, 455–464. https://doi.org/10.1109/ICWS60048.2023.00064 ISSN: 2836-3868.

[S67] Yuhanis Yusof and Omer F. Rana. 2010. Classification of Software Artifacts Based on Structural Information. In *Knowledge-Based and Intelligent Information and Engineering Systems*, Rossitza Setchi, Ivan Jordanov, Robert J. Howlett, and Lakhmi C. Jain (Eds.). Springer, Berlin, Heidelberg, 546–555. https://doi.org/10.1007/978-3-642-15384-6_58

[S68] Lingxiao Zhang, Yanzhen Zou, Bing Xie, and Zixiao Zhu. 2014. Recommending relevant projects via user behaviour: an exploratory study on github. In *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies (CrowdSoft 2014)*. Association for Computing Machinery, New York, NY, USA, 25–30. https://doi.org/10.1145/2666539.2666570

[S69] Yu Zhang, Frank F. Xu, Sha Li, Yu Meng, Xuan Wang, Qi Li, and Jiawei Han. 2019. HiGitClass: Keyword-Driven Hierarchical Classification of GitHub Repositories. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, Beijing, China, 876–885. https://doi.org/10.1109/ICDM.2019.00098 ISSN: 2374-8486.

[S70] Yuming Zhou and Baowen Xu. 2008. Predicting the maintainability of open source software using design metrics. *Wuhan University Journal of Natural Sciences* 13, 1 (Feb. 2008), 14–20. https://doi.org/10.1007/s11859-008-0104-6