RESEARCH-ARTICLE

Automatic Categorization of GitHub Actions with Transformers and Few-shot Learning

PHUONG T. NGUYEN, University of L'Aquila, L'Aquila, AQ, Italy

I work with Machine Learning techniques and their applications in Software Engineering.

JURI DI ROCCO, University of L'Aquila, L'Aquila, AQ, Italy

CLAUDIO DI SIPIO, University of L'Aquila, L'Aquila, AQ, Italy

MUDITA SHAKYA, University of L'Aquila, L'Aquila, AQ, Italy

DAVIDE DI RUSCIO, University of L'Aquila, L'Aquila, AQ, Italy

MASSIMILIANO DI PENTA, University of Sannio, Benevento, BN, Italy

Published: 24 October 2024

Citation in BibTeX format

ESEM '24: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement
October 24 - 25, 2024
Barcelona, Spain

Conference Sponsors:
SIGSOFT

ESEM '24: Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (October 2024)
https://doi.org/10.1145/3674805.3690752
ISBN: 9798400710476

Automatic Categorization of GitHub Actions with Transformers and Few-shot Learning

Phuong T. Nguyen
DISIM, University of L'Aquila
Italy
phuong.nguyen@univaq.it

Juri Di Rocco
DISIM, University of L'Aquila
Italy
juri.dirocco@univaq.it

Claudio Di Sipio
DISIM, University of L'Aquila
Italy
claudio.disipio@univaq.it

Mudita Shakya
University of L'Aquila
Italy
mudita.shakya@student.univaq.it

Davide Di Ruscio
DISIM, Università degli Studi dell'Aquila
Italy
davide.diruscio@univaq.it

Massimiliano Di Penta
University of Sannio
Italy
dipenta@unisannio.it

Abstract

In the GitHub ecosystem, workflows are used as an effective means to automate development tasks and to set up a Continuous Integration and Delivery (CI/CD pipeline). `GitHub Actions` (GHA) has been conceived to provide developers with a practical tool to create and maintain workflows, avoiding "reinventing the wheel" and cluttering the workflow with shell commands. Properly leveraging the power of `GitHub Actions` can facilitate the development processes, enhance collaboration, and significantly impact project outcomes. To expose actions to search engines, GitHub allows developers to assign them to one or more categories manually. These are used as an effective means to group actions sharing similar functionality. Nevertheless, while providing a practical way to execute workflows, many actions have unclear purposes, and sometimes they are not categorized. In this work, we bridge such a gap by conceptualizing `Gavel`, a practical solution to increasing the visibility of actions in GitHub. By leveraging the content of `README.MD` files for each action, we use Transformer to assign suitable categories to the action. We conducted an empirical investigation and compared `Gavel` with a state-of-the-art baseline. The results show that our approach can assign categories to GitHub actions effectively, thus outperforming the baseline.

Keywords

GitHub Actions, Pre-trained models, Sentence transformers, Few-shot learning

ACM Reference Format:

Phuong T. Nguyen, Juri Di Rocco, Claudio Di Sipio, Mudita Shakya, Davide Di Ruscio, and Massimiliano Di Penta. 2024. Automatic Categorization of GitHub Actions with Transformers and Few-shot Learning. In *Proceedings of the 18th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), October 24–25, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3674805.3690752

ESEM '24, October 24–25, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-1047-6/24/10
https://doi.org/10.1145/3674805.3690752

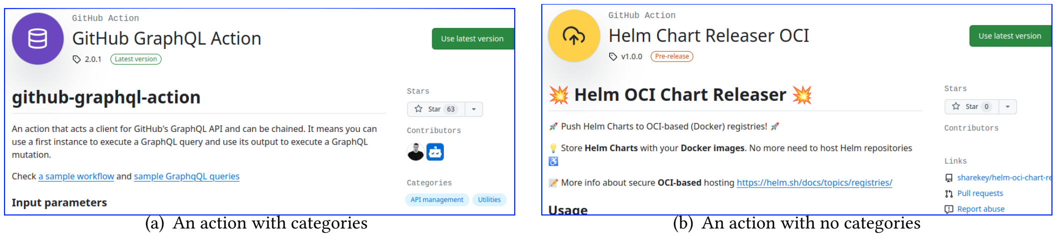
1 Introduction

Within a GitHub repository, a workflow consists of a series of automated steps executed in response to specific events. Workflows can be initiated through triggering events, including code commits, issues, pull requests, comments, or scheduled events, to name a few [17]. This facilitates the automation of various aspects of software development and project management. In fact, workflows simplify the automation of specific tasks (e.g., configuring a particular environment, releasing software on a package registry, running tests, or assessing code quality) without the need to write custom code for each task. In GitHub, workflows are the premier way to automate CI/CD (Continuous Integration and Delivery) [23]. Actions are reusable assets that can be used in place of custom scripts and are available in public GitHub repositories and a specific GitHub Marketplace.[1]

When uploading an action to the Marketplace, developers usually assign it to some *categories* to characterize its constituent functionality. These categories are an effective means to group similar actions into common classes, allowing other developers to approach the ones that match their needs. This fosters reusability among users in the open-source community [8], providing them with artefacts relevant to the considered tasks. Nevertheless, while offering a practical way to ease the development activities, many actions are not visible to users, i.e., several of them are not specified to any categories, being obscure to search engines. Thus, it would be important to support developers in effectively defining categories for the GitHub actions they publish in Marketplace.

By means of a lightweight systematic literature review (SLR), in this work we found out that no Software Engineering (SE) work has ever been conducted to automatically attach categories to GitHub actions. Being motivated by this gap, we conceive `Gavel`,[2] a GitHub actions visibility elevator based on transformers and few-shot learning to make actions more visible and accessible to developers. Given an action, we use its `README.MD` as input data, and build a multi-label classifier combined with few-shot

[1] https://github.com/marketplace/
[2] Gavel is a hammer usually made of wood, and used in events like a court, or an auction to attract attention from the audience.



(a) An action with categories (b) An action with no categories

Figure 1: Examples of GitHub actions with and without categories.

learning to assign one or more categories to it. We evaluate the performance of Gavel using a dataset collected from GitHub, and compare it with a quasi-equivalent tool. The experiments show that Gavel is effective at assigning categories to actions, obtaining an encouraging prediction, thereby outperforming the baseline.

The contributions of our work are summarized as follows:

- An approach named Gavel built on top of sentence transformers to equip GitHub actions with proper categories to increase their visibility.
- An empirical evaluation of Gavel on real-world datasets, and comparison with a state-of-the-art baseline.
- The tool together with the data curated in this work is published online to enable future research [18].

2 Motivation and background

This section takes a concrete example to highlight the importance of categories for GitHub actions, and reviews the related studies.

2.1 Motivating example

Figure 1 displays a motivating example with 2 actions. The first one named `GitHub GraphQL Action`[3] (see Figure 1(a)) is employed in executing a GraphQL query, and using its output to run a GraphQL mutation. The action has been classified into two categories, i.e., `API management` and `Utilities`. When we expand one of the categories, we will be redirected to a set of all the actions belonging to the same category. This means that whenever users look into either the `API management` or the `Utilities` category, they will see `GitHub GraphQL Action` among the entries. The action in Figure 1(b) with the name `Helm Chart Releaser OCI`[4] can be used to push Helm Chart to container registries, including those hosted in GitHub or Azure. Noteworthy, this action has not been sorted into any categories, and due to this, it will never show up in the search towards the available categories in Marketplace. Essentially, this limits the action visibility and, consequently, its adoption.

The example implies the need for a proper technique to recommend categories for actions. We conjecture that the `README.MD` file(s) of an action can be used as input for a recommender of categories. This requires suitable categorization algorithms able to perform multi-label classification.

2.2 Related Work

Prana et al. [20] conducted a qualitative study to manually label the content of 393 randomly selected `README.MD` files. They identified eight categories in the coding schema coming from a qualitative analysis, and reported the respective frequencies and associations. They also developed an approach to the categorization of `README.MD` files. GHTRec [29] predicts the list of topics using the preprocessed content using the BERT model. Furthermore, the tool suggests the most similar trending repositories, namely the most starred ones, by computing the similarity on the topic vectors mined from the user's historical data.

Izadi et al. [11] leveraged a fine-tuned version of BERT [6] to classify GitHub repositories using textual content. The training data includes 152K GitHub repositories and 228 featured topics. Then, the tool exploits a fine-tuned version of DistilBERT to classify a repository from all the collected textual data, i.e., `README.MD` content, project name, descriptions, wiki pages, and file names. The same authors enriched the classifier by exploiting semantic relationships [12]. To this end, a knowledge graph composed of 2,234 semantic relationships and 863 curated topics was built according to *(i)* their popularity and *(ii)* the degree in the produced graph. The novel approach, called SED-KG, is employed to enhance the performance of existing topic recommender systems.

HybridRec [7] predicts a list of topics for a given GitHub repository by relying on 2 components, i.e., a stochastic classifier and a collaborative filtering engine. The former is a Complement Naïve Bayesian network (`CNBN`) to encode `README.MD` content using TF-IDF, predicting an initial set of topics. The latter exploits such predictions to infer additional tags, thus increasing the coverage by considering the mutual relationships encoded as a graph. GitRanking [24] uses the active sampling algorithm for Pairwise comparisons (ASAP) to build a topic taxonomy by combining `README.MD` files and Wikidata definition. A manual filtering is applied by involving real developers in annotating the most 3,000 popular topics. Then, ASAP is used to perform the final ranking by leveraging the WikiData description of each topic.

Mastropaolo et al. [17] presented GH-WCOM (GitHub Workflow COMpletion), a Transformer-based approach to assist developers in specifying GitHub workflows. The authors also conducted a qualitative analysis to investigate the extent to which the recommendations provided by GH-WCOM are useful when the generated output is different from the target one. We conjecture that Gavel can be complementary to GH-WCOM, and possibly improve its action suggestions.

By reviewing the related studies, we see that while relevant to our work in terms of tasks, the aforementioned pieces of work deal with the classification of other types of artifacts—such as GitHub repositories—rather than GitHub actions, and none of them has tackled the lack of training data. Among others, only `CNBN` [7] and

[3] https://github.com/marketplace/actions/github-graphql-action
[4] https://github.com/marketplace/actions/helm-chart-releaser-oci

Repository Catalogue [11, 12] work with multi-label classification of artifacts, albeit not GitHub actions. Through an evaluation, we found out that they treat a `README.MD` file as a whole, without distinguishing different elements such as code, or text, failing to categorize GitHub actions. This triggers the need for a tool to assign labels to actions, that should be able to handle diverse components of `README.MD` files, as well as to rectify the lack of labeled data.

3 Proposed Approach

This section describes in detail `Gavel`–our conceived tool for increasing the visibility of GitHub actions built on top of transformers and few-shot learning (see Figure 2). Given an action, using the `README.MD` file(s) as the sole input, `Gavel` generates a list of possible categories to which the action should belong.

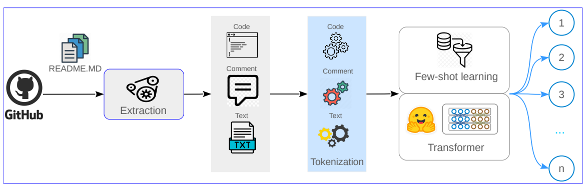


Figure 2: System architecture.

3.1 Extraction and Tokenization

In Figure 2, the first phase involves parsing data from raw `README.MD` files. To extract the needed data to separate fields, i.e., code, comments, and text from files for the classification task, we used a dedicated Python library for *.MD* files, named Marko.[5] Compared to existing libraries [1, 27], it is compliant with the latest *CommonMark* specifications,[6] thus improving the overall parsing.

The resulting data is then encoded using a tokenizer, generating a numerical representation that captures the meaning of the input. Tokenization is a crucial pre-processing step to prepare data for DL models. By breaking down the data into the constituent parts, containing words or sub-words called tokens, the tokenization enables models to learn from the structure and semantics of the data, leading to better performance for various tasks [9].

Listing 1: Example of code and comments.

```
# This function greets the user
def greet(name):
print(f"Hello, {name}!")
greet("Alice")
```

Listing 1 depicts a simple code snippet, together with code comments. The transformed sequences for comments and code are: `BOS, #, This, function, greets, the, user, EOS` and `BOS, def, greet, (, name, ), :, print, (, f, ", Hello, , name! "), greet, (", Alice, "), EOS`. BOS and EOS are two special delimiters to signify the beginning and end of a sequence.

3.2 Multi-label Classification

Once the data has been collected and encoded with the tokenizer, it is necessary to parse them with a suitable format. The input contains X as a set of m `README.MD` files, together with matrix $Y[m,n]$ to store the labels, i.e., categories. Each row specifies the categories of a repository: if the repository belongs to a category, then the corresponding column is set to 1, otherwise it is set to 0. The pseudo-code in Algorithm 1 illustrates how the process unfolds. Starting from these matrices, we train a set of classifiers, adopting the one vs. rest classification paradigm [19]. For each class, a classifier assigns each sample to one or more categories, which are not mutually exclusive. Due to space limit, we cannot recall the algorithm here. Interested readers are kindly referred to the work of Pawara et al. [19] for details.

Algorithm 1: One vs. all multi-label classification

Input :
- X: m `README.MD` files containing text, code, comments.
- Y: Matrix of categories of size $[m, n]$.

Output:
- $C = [c_1..c_n]$: A set of n binary classifiers, each corresponds to one category.

$C \leftarrow \emptyset$
foreach $i \in [1..n]$ **do**
  $c_i \leftarrow \emptyset$
  $y_i \leftarrow Y[:, i]$
  $c_i \leftarrow Train(X, y_i)$
  $C \leftarrow C \bigcup c_i$
end
return C

3.3 Few-shot learning

Deep learning models usually require a large amount of data for training [22]. To feed supervised learning algorithms, data needs to be properly labeled, which is usually done by humans. In fact, collecting data is a error-prone and daunting task. To this end, few-shot learning has been proposed as a practical solution to the lack of training data [2], allowing a model to be trained with a small amount of data. For each class, only a few samples are selected to train the model, and the rest is used for fine tuning. This simulates the case where only a limited amount of data is available. Such a model aims to learn better by using relevant, and well-curated samples, rather than low-quality data [26]. Moreover, training or fine-tuning with less data helps save time and resources.

For the implementation of `Gavel`, we make use of `SetFit`,[7] a pre-trained transformer model of text hosted in Hugging Face. Since the fine-tuning is resource demanding, `Gavel` was trained and tested on a Linux High-Performance Computing (HPC) cluster. Instead, `CNBN`[8] is more lightweight, and it was run on a local machine with OS Windows 11 Processor AMD Ryzen 9 RAM 16 GB.

4 Evaluation Methodology

This section describes in detail the materials and methods used to evaluate `Gavel`, and compare it with a state-of-the-art baseline.

[5] https://marko-py.readthedocs.io/en/latest/index.html
[6] https://spec.commonmark.org/0.30/
[7] https://huggingface.co/docs/setfit/index
[8] We use the original implementation available at: https://github.com/MDEGroup/HybridRec

4.1 Research Questions

▹ **RQ$_1$**: *Has the issue of categorizing GitHub actions been investigated by Software Engineering research?* In this research question, we perform a lightweight systematic literature review to investigate to what extent the issue of categorization of GitHub actions has been studied by state-of-the-art research.

▹ **RQ$_2$**: *Which configuration brings about the best prediction accuracy?* `Gavel` can accept as input different types of `README.MD` data, including text, code, or code together with comments, or all of them. We evaluate which of these configurations is more effective, bringing a superior performance with respect to accuracy.

▹ **RQ$_3$**: *How does* `Gavel` *compare with* `CNBN`*?* To the best of our knowledge, no work has ever been conducted to categorizing GitHub actions, thus for comparison, in this work we have to consider `CNBN` [7], a baseline that has been conceived for a different domain, i.e., categorization of GitHub topics.

4.2 Metrics and Datasets

▹ *Metrics.* There are the following definitions: True positives (TP): the number of correctly predicted categories; False positives (FP): the number of incorrectly predicted categories; False negatives: the number of incorrectly not predicted categories. The metrics are defined as follows [3]: Precision $P = TP/(TP + FP)$; Recall $R = TP/(TP + FN)$; and F1-score $F1 = 2 * P * R/(P + R)$. For the presentation, we use the micro average, macro average, and weighted average score of these metrics.

▹ *Dataset.* As GitHub does not provide an API to retrieve actions from Marketplace, we had to collect data from: *(i)* a dataset [5] consists of 958 actions; and *(ii)* the second one [14] includes 708 actions. We combined them to create a single dataset. This is done by aligning the actions' metadata, resolving discrepancies, and removing duplicates. The final dataset contains 1,213 unique actions, spanning in 30 categories, and covering diverse functionality.

4.3 Configurations

The `README.MD` file(s) of an action may contain different types of data, including text, code, and comments. We consider the following regular data types: *(i) Text.* Most `README.MD` files contain text to describe the main functionalities of actions; *(ii) Source code.* `README.MD` files usually include source code as a means to illustrate how a workflow can be executed; and *(iii) Code comments.* They are embedded text to describe the corresponding source code.

Table 1: Experimental configurations.

Conf.	Text	Code	Comments	# of samples
C_1	✔	✔	✔	1,213
C_2	✗	✔	✔	1,140
C_3	✗	✗	✔	436
C_4	✗	✔	✗	1,140
C_5	✔	✗	✗	1,210

Table 1 lists the configurations considered in our experiments. By C_1 all data, including code, comments, and text in `README.MD` is used as input, while with C_2, text is not considered. By C_3, comments in code are extracted and used as the sole input. Code is the only artifact considered as input by C_4. Eventually, in C_5, only textual data in `README.MD` is parsed and fed to train `Gavel`. Depending on the availability of data, many files do not include some of the aforementioned artifacts. This is the reason why the number of samples–depicted in the last column of the table–varies by the configurations. For instance, there are only 436 samples for C_3 as few `README.MD` files have comments embedded in code.

5 Results and Discussion

This section reports the results by answering the research questions.

5.1 RQ$_1$: *Has the issue of categorizing GitHub actions been investigated by Software Engineering research?*

We conducted a lightweight SLR [16] inspired by the well-established *"four W-question strategy"* [28] to select studies that deal with the categorization of GitHub artifacts, explained as follows.

▹ *Which?* We ran a comprehensive search, combining both automated and manual methods to retrieve relevant work. The search string is as follows: `(workflow* OR action* OR CI/CD OR continuous integration OR DevOps OR CICD OR YAML) AND (automat* OR classification OR recommend* OR complet* OR categorization OR tag*) AND (GitHub)`.

▹ *Where?* Our analysis focused on prominent SE venues, encompassing 9 conferences: ASE, ESEC/FSE, ESEM, ICSE, ICSME, ICST, ISSTA, MSR, SANER, and 5 journals: EMSE, IST, JSS, TOSEM, and TSE [18]. We accessed *Scopus*[9] and employed the advanced search and export functions to retrieve all papers published in the aforementioned venues within the given temporal range. A tailored Python script was used to filter out irrelevant papers [18].

▹ *What?* For each article, we extracted information from the title and abstract by applying predefined keywords to ensure relevance to our research focus.

▹ *When?* As the introduction of GitHub workflows is very recent, our search was limited to the most recent five years, i.e., from 2017 to 2023. This allows us to capture the latest developments. The query was executed in May 2024, hence 2024 was not considered.

Running the query, we obtained 48 papers that matched the keywords. Afterward, we carefully read the title and abstract, and identified seven studies relevant to the topic under consideration. We filtered out two papers since they analyze generic CI/CD tools, ending up with five papers, summarized as follows.

Valenzuela-Toledo et al. [25] performed a manual categorization of GHA using the card sorting methodology [13]. To this end, a dataset of 222 commits and 10 open-source projects has been collected to investigate file modifications. The authors eventually report 11 types of modification in workflows, including primary and secondary groups. The findings demonstrate that developers need more support in editing operations of existing workflow.

Calefato et al. [4] investigated the usage of MLOps operations in workflows. A dataset of 337 valid workflows was extracted from existing work. The authors selected workflows using at least an ML model in the pipeline. After the data collection phase, the dataset is used to search for the most frequent MLOps operations employed.

Golzadeh et al. [10] carried out a large-scale empirical study on 8 popular CI services, including GHA. By using an open-source npm

[9] https://scopus.com

Table 2: Classification report for C_1, C_2, C_3, and C_4.

	C_1				C_2				C_3				C_4			
Category	P	R	F1	#	P	R	F1	#	P	R	F1	#	P	R	F1	#
AI Assisted	1.00	0.20	0.33	5	1.00	0.20	0.33	5	1.00	1.00	1.00	1	1.00	0.20	0.33	5
API management	1.00	0.07	0.13	14	0.50	0.07	0.12	14	1.00	1.00	1.00	1	1.00	0.07	0.13	14
Backup Utilities	1.00	0.50	0.67	2	1.00	0.50	0.67	2	1.00	1.00	1.00	1	1.00	0.50	0.67	2
Chat	0.72	0.57	0.64	37	0.84	0.57	0.68	37	1.00	1.00	1.00	1	0.83	0.65	0.73	37
Code Scanning Ready	1.00	0.33	0.50	3	1.00	0.33	0.50	3	1.00	1.00	1.00	1	1.00	0.33	0.50	3
Code quality	0.76	0.84	0.80	129	0.75	0.81	0.78	123	0.44	0.88	0.58	8	0.76	0.81	0.78	123
Code review	0.75	0.67	0.71	103	0.71	0.65	0.68	97	0.50	0.25	0.33	8	0.74	0.69	0.71	97
Code search	1.00	0.12	0.22	8	1.00	0.17	0.29	6	1.00	1.00	1.00	1	1.00	0.17	0.29	6
Community	1.00	0.08	0.15	25	0.50	0.05	0.08	22	1.00	1.00	1.00	1	1.00	0.09	0.17	22
Container CI	0.56	0.15	0.24	60	0.50	0.09	0.15	56	1.00	0.25	0.40	4	0.59	0.18	0.27	56
Continuous integration	0.86	0.95	0.90	434	0.89	0.94	0.91	407	0.44	0.67	0.53	30	0.87	0.95	0.91	407
Dependency management	0.88	0.65	0.75	89	0.75	0.57	0.65	82	0.10	0.33	0.15	3	0.73	0.59	0.65	82
Deployment	0.75	0.84	0.79	194	0.77	0.81	0.79	189	0.70	0.33	0.45	21	0.75	0.84	0.79	189
Desktop tools	1.00	0.10	0.18	10	1.00	0.10	0.18	10	1.00	1.00	1.00	1	1.00	0.10	0.18	10
GitHub Sponsors	1.00	0.50	0.67	2	1.00	0.50	0.67	2	1.00	1.00	1.00	1	1.00	0.50	0.67	2
IDEs	1.00	0.17	0.29	6	1.00	0.17	0.29	6	1.00	1.00	1.00	1	1.00	0.17	0.29	6
Learning	1.00	0.33	0.50	3	1.00	0.33	0.50	3	1.00	1.00	1.00	1	1.00	0.33	0.50	3
Localization	1.00	0.25	0.40	4	1.00	0.33	0.50	3	1.00	0.50	0.67	2	1.00	0.33	0.50	3
Mobile	1.00	0.20	0.33	5	1.00	0.20	0.33	5	1.00	1.00	1.00	1	1.00	0.20	0.33	5
Mobile CI	0.67	0.22	0.33	9	1.00	0.25	0.40	8	1.00	1.00	1.00	1	1.00	0.12	0.22	8
Monitoring	1.00	0.07	0.12	15	1.00	0.07	0.13	14	1.00	1.00	1.00	1	0.50	0.07	0.12	14
Open Source management	0.56	0.29	0.39	78	0.51	0.32	0.39	75	0.25	0.20	0.22	5	0.55	0.29	0.38	75
Project management	0.67	0.38	0.48	77	0.60	0.40	0.48	75	0.20	0.25	0.22	4	0.62	0.32	0.42	75
Publishing	0.64	0.46	0.54	149	0.74	0.54	0.62	145	0.50	0.21	0.30	19	0.67	0.54	0.60	145
Reporting	0.53	0.44	0.48	36	0.69	0.50	0.58	36	1.00	0.50	0.67	2	0.68	0.53	0.59	36
Security	0.62	0.35	0.45	37	0.67	0.35	0.46	34	1.00	0.33	0.50	3	0.60	0.35	0.44	34
Support	1.00	0.08	0.14	13	1.00	0.08	0.15	12	1.00	1.00	1.00	1	1.00	0.08	0.15	12
Testing	0.80	0.77	0.79	102	0.80	0.80	0.80	100	0.50	0.20	0.29	5	0.89	0.80	0.84	100
Time tracking	1.00	0.50	0.67	2	1.00	0.50	0.67	2	1.00	1.00	1.00	1	1.00	0.50	0.67	2
Utilities	0.85	0.91	0.88	402	0.86	0.92	0.89	383	0.38	0.48	0.43	31	0.86	0.91	0.88	383
micro avg	0.79	0.71	0.75	2,053	0.80	0.71	0.75	1,956	0.48	0.49	0.49	161	0.80	0.71	0.75	1,956
macro avg	0.85	0.40	0.48	2,053	0.84	0.40	0.49	1,956	0.80	0.71	0.72	161	0.85	0.41	0.49	1,956
weighted avg	0.79	0.71	0.72	2,053	0.79	0.71	0.72	1,956	0.55	0.49	0.48	161	0.79	0.71	0.72	1,956
avg	0.72	0.62	0.64	2,053	0.72	0.62	0.65	1,956	0.39	0.33	0.34	161	0.73	0.63	0.65	1,956

API, 91,810 different GitHub repositories were analyzed in terms of CI evolution, combination, and popularity. The results show that usage of GitHub actions in the CI/CD pipeline is increasing since they provide an easy-to-use integration mechanism. The same authors ran a qualitative study [21] by interviewing 22 experienced software engineers in CI development. The study paves the way for reusable workflows using GHA. Similarly, Kinsman et al. [15] investigated how developers use GHA to automate various tasks.

> ***Answer to RQ$_1$:*** The lightweight SLR reveals that the usage of GitHub actions is still in its infancy, and no automated technique to classify actions into categories has been proposed yet, triggering the need for a suitable tool.

5.2 RQ$_2$: *Which configuration brings about the best prediction accuracy?*

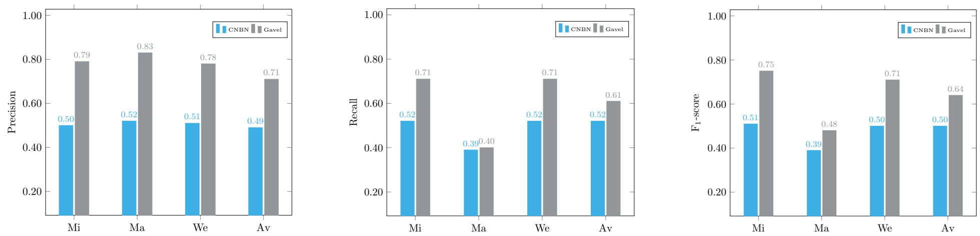
We experimented `Gavel` with configurations in Table 1. By carefully investigating the results, we noticed that by C_5, `Gavel` gets a comparable performance compared to that when using C_1, C_2, and C_4, thus we do not depict the results for C_5 due to space limits, but report them later in the comparison with `CNBN` in RQ$_3$.

In Table 2, we show Precision, Recall, F_1 score together with the number of samples (#). It is evident that the categories are considerably imbalanced, i.e., while some of them have a large number of samples, most of the categories are sparse. Investigating the table, we see that `Gavel` obtains perfect precision by various categories. However, the recall scores are low in some of the categories. While high precision means that the recommendations are mostly relevant and accurate, low recalls indicate that the system fails to recommend many of the relevant items. With C_1, C_2, and C_4, the performance of `Gavel` is good by categories in which there is a large number of samples. For example, by `Utilities` where there are more than 400 samples (C_1), the precision, recall, and F1 scores are 0.85, 0.91, and 0.88, respectively. With C_3, `Gavel` performs worse by various categories, e.g., F1 is 0.45 by `Deployment`. However, also with C_3, `Gavel` gets a good prediction by sparse categories. We count 16 among 30 categories by which there is just one testing sample, e.g., `Desktop tools`, `GitHub Sponsors`, `IDEs`, but `Gavel` properly classifies them. When summing up with the micro, macro, and weighted average values, we see that the accuracy obtained with C_3 is very low, i.e., only macro average is greater than 0.70, and the remaining scores are lower than 0.50.

> ***Answer to RQ$_2$:*** When only code comments are considered as input, `Gavel` obtains the worst performance. Meanwhile, its prediction is comparable by the remaining configurations.

5.3 RQ$_3$: *How does* `Gavel` *compare with* `CNBN`*?*

RQ$_1$ shows that there has been no approach conceived to the categorization of GitHub actions. Therefore, we have to find a comparable approach for benchmarking `Gavel`. By investigating the related work in Section 2.2, we noticed that `CNBN` [7] is a state-of-the-art tool that tackles a similar issue, i.e., supporting multi-label categorization of generic GitHub projects using `README.MD` files.



(a) Precision (b) Recall (c) F_1 score

Figure 3: Comparison of accuracy between CNBN and Gavel.

To avoid any bias, we select the best configuration of Gavel as well as of CNBN, i.e., with Gavel we used the results obtained for C_5, corresponding to a slightly better accuracy compared with the remaining configurations. We set the CNBN cutoff value to 2, since it represents the median value of the considered GHA dataset.

Figure 3 depicts the comparison between Gavel and CNBN in terms of Precision, Recall, and F1 score. For each metric, we report its micro average (Mi), macro average (Ma), weighted average (We), and Average (Av) scores. Notably, Gavel outperforms CNBN in all the metrics, e.g., concerning Precision, the corresponding scores for CNBN are always lower than 0.53, while Gavel obtains much higher values, greater than 0.70. The same can also be seen with the other metrics. Altogether, it is evident that our proposed approach yields a substantially superior accuracy compared to that by CNBN.

Besides accuracy, we also measured the time for the training and testing phases of the two models. Due to space limits, we cannot report the detailed measurement here, but give a brief summary as follows. By examining the results, we can see that compared to Gavel, CNBN is much faster both in training and testing. On average, the CNBN model requires around one second for the training phase and less than one second for each fold to perform the testing phase. Gavel requires more time for both phases, i.e., several hundreds seconds. In this respect, CNBN can be seen as a valuable alternative when there is no hardware resource to fine-tune Gavel.

Answer to RQ$_3$: Gavel substantially outperforms CNBN in terms of precision, recall, and F1-score. However, CNBN is more timing efficient compared to Gavel, complement to Gavel when no powerful platform is available for training.

5.4 Limitations

Though Gavel was conceived to work with different tokenizers, in the scope of this paper, we managed to apply a common tokenizer to encode all types of data (code, comments, text). In fact, using a sole tokenizer cannot capture all the intrinsic features in each data type. This can be considered as a limitation of our work, leaving room for improvement. We conjecture that tokenizers tailored to each type of input data will help the tool capture the intrinsic features, thus further improving its ability to provide suitable categories.

Gavel obtains a better prediction accuracy compared to that of CNBN. However, it suffers from prolonged fine-tuning, though being trained with few-shot samples. We see that while pre-trained models are effective, they come at a price in terms of training resources. Thus, conventional models like CNBN can be seen as a valuable alternative when no powerful platforms are available for fine-tuning the Gavel model. In this respect, it is necessary to look for more lightweight techniques—compared to transformer—that can maintain a trade-off between accuracy and effectiveness.

5.5 Threats to Validity

▹ *Internal validity.* This is related to the fact that our evaluation might not resemble real-world scenarios, i.e., some categories are supported by few samples. To mitigate this, we curated the data collection by retrieving information from different sources. The SLR may not cover all relevant work, to reduce the threat, we considered prominent SE venues and carefully checked the titles and abstracts.

▹ *External validity.* This concerns the generalizability of the findings outside this study, i.e., Gavel may not perform similarly if applied on different datasets. To mitigate this, we experimented with five different configurations considering various kinds of data, i.e., plain text, code, or mixed content.

6 Conclusion and Future work

We presented Gavel as a practical approach to the categorization of GitHub actions. Through a literature review conducted on premier SE venues, we realized that this is the first tool dealing with this issue. An empirical evaluation on a dataset collected from GitHub Marketplace demonstrated that our proposed approach can provide suitable categories for actions, and it gains a better performance compared to a state-of-the-art benchmark. For future work, we plan to equip Gavel with lightweight classification models, aiming to maintain a trade-off between effectiveness and efficiency.

Acknowledgments

This work has been partially supported by the EMELIOT research project, which has been funded by the MUR under the PRIN 2020 program grant n. 2020W3A5FY. The work has been also partially supported by the European Union–NextGenerationEU through the Italian Ministry of University and Research, Projects PRIN 2022 PNRR *"FRINGE: context-aware FaiRness engineerING in complex software systEms"* grant n. P2022553SL. We acknowledge the Italian "PRIN 2022" project TRex-SE: *"Trustworthy Recommenders for Software Engineers,"* grant n. 2022LKJWHC.

References

[1] 2023. Python-Markdown. https://github.com/Python-Markdown/markdown original-date: 2010-05-29T02:59:45Z.

[2] Sebastian Blaes and Thomas Burwick. 2017. Few-shot learning in deep networks through global prototyping. *Neural Networks* 94 (2017), 159–172. https://doi.org/10.1016/j.neunet.2017.07.001

[3] Michael Buckland and Fredric Gey. 1994. The relationship between recall and precision. *Journal of the American society for information science* 45, 1 (1994), 12–19. Publisher: Wiley Online Library.

[4] Fabio Calefato, Filippo Lanubile, and Luigi Quaranta. 2022. A Preliminary Investigation of MLOps Practices in GitHub. In *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '22)*. Association for Computing Machinery, New York, NY, USA, 283–288. https://doi.org/10.1145/3544902.3546636

[5] Alexandre Decan, Tom Mens, and Hassan Onsori Delicheh. 2023. On the outdatedness of workflows in the GitHub Actions ecosystem. *Journal of Systems and Software* 206 (2023), 111827. https://doi.org/10.1016/j.jss.2023.111827

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/ARXIV.1810.04805

[7] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T. Nguyen, and Riccardo Rubei. 2022. HybridRec: A recommender system for tagging GitHub repositories. *Applied Intelligence* (Aug. 2022). https://doi.org/10.1007/s10489-022-03864-y

[8] Claudio Di Sipio, Riccardo Rubei, Juri Di Rocco, Davide Di Ruscio, and Phuong T. Nguyen. 2024. Automated categorization of pre-trained models in software engineering: A case study with a Hugging Face dataset. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (Salerno, Italy) *(EASE '24)*. Association for Computing Machinery, New York, NY, USA, 351–356. https://doi.org/10.1145/3661167.3661215

[9] Kexue Fu, Peng Gao, Shaolei Liu, Linhao Qu, Longxiang Gao, and Manning Wang. 2024. POS-BERT: Point cloud one-stage BERT pre-training. *Expert Systems with Applications* 240 (2024), 122563. https://doi.org/10.1016/j.eswa.2023.122563

[10] Mehdi Golzadeh, Alexandre Decan, and Tom Mens. 2022. On the rise and fall of CI services in GitHub. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 662–672. https://doi.org/10.1109/SANER53432.2022.00084 ISSN: 1534-5351.

[11] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. 2021. Topic recommendation for software repositories using multi-label classification algorithms. *Empirical Software Engineering* 26, 5 (2021), 93. https://doi.org/10.1007/s10664-021-09976-2

[12] 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

[13] Christos Katsanos, Vasileios Christoforidis, and Christina Demertzi. 2023. Task-Based Open Card Sorting: Towards a New Method to Produce Usable Information Architectures. In *Human Interface and the Management of Information*, Hirohiko Mori and Yumi Asahi (Eds.). Springer Nature Switzerland, Cham, 68–80.

[14] Timothy Kinsman, Mairieli Wessel, Marco A. Gerosa, and Christoph Treude. 2021. How Do Software Developers Use GitHub Actions to Automate Their Workflows?. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 420–431. https://doi.org/10.1109/MSR52588.2021.00054

[15] Timothy Kinsman, Mairieli Wessel, Marco A. Gerosa, and Christoph Treude. 2021. How Do Software Developers Use GitHub Actions to Automate Their Workflows? IEEE Computer Society, 420–431. https://doi.org/10.1109/MSR52588.2021.00054

[16] 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 (2009), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009 Special Section - Most Cited Articles in 2002 and Regular Research Papers.

[17] Antonio Mastropaolo, Fiorella Zampetti, Gabriele Bavota, and Massimiliano Di Penta. 2024. Toward Automatically Completing GitHub Workflows. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 13, 12 pages. https://doi.org/10.1145/3597503.3623351

[18] Phuong T. Nguyen, Juri Di Rocco, Claudio Di Sipio, Mudita Shakya, Davide Di Ruscio, and Massimiliano Di Penta. 2024. *Replication Package: Automatic Categorization of GitHub Actions with Transformers and Few-shot Learning*. https://anonymous.4open.science/r/Gavel-Replication-Package-5786/

[19] Pornntiwa Pawara, Emmanuel Okafor, Marc Groefsema, Sheng He, Lambert R.B. Schomaker, and Marco A. Wiering. 2020. One-vs-One classification for deep neural networks. *Pattern Recognition* 108 (2020), 107528. https://doi.org/10.1016/j.patcog.2020.107528

[20] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the Content of GitHub README Files. *Empir. Softw. Eng.* 24, 3 (2019), 1296–1327. https://doi.org/10.1007/s10664-018-9660-3

[21] Pooya Rostami Mazrae, Tom Mens, Mehdi Golzadeh, and Alexandre Decan. 2023. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. *Empirical Software Engineering* 28, 2 (March 2023), 52. https://doi.org/10.1007/s10664-022-10285-5

[22] Iqbal H. Sarker. 2021. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Comput. Sci.* 2, 6 (2021), 420. https://doi.org/10.1007/S42979-021-00815-1

[23] Sk Golam Saroar and Maleknaz Nayebi. 2023. Developers' Perception of GitHub Actions: A Survey Analysis. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering* (Oulu, Finland) *(EASE '23)*. Association for Computing Machinery, New York, NY, USA, 121–130. https://doi.org/10.1145/3593434.3593475

[24] 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 (2023), 1982–2006. https://doi.org/10.1002/spe.3238 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3238.

[25] Pablo Valenzuela-Toledo and Alexandre Bergel. 2022. Evolution of GitHub Action Workflows. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 123–127. https://doi.org/10.1109/SANER53432.2022.00026 ISSN: 1534-5351.

[26] Leiming Yan, Yuhui Zheng, and Jie Cao. 2018. Few-shot learning for short text classification. *Multim. Tools Appl.* 77, 22 (2018), 29799–29810. https://doi.org/10.1007/S11042-018-5772-4

[27] Hsiaoming Yang. 2023. Mistune v3. https://github.com/lepture/mistune original-date: 2014-02-18T10:13:04Z.

[28] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying Relevant Studies in Software Engineering. *Inf. Softw. Technol.* 53, 6 (jun 2011), 625–637. https://doi.org/10.1016/j.infsof.2010.12.010

[29] Yuqi Zhou, Jiawei Wu, and Yanchun Sun. 2021. GHTRec: A Personalized Service to Recommend GitHub Trending Repositories for Developers. In *2021 IEEE International Conference on Web Services, ICWS 2021, Chicago, IL, USA, September 5-10, 2021*, Carl K. Chang, Ernesto Daminai, Jing Fan, Parisa Ghodous, Michael Maximilien, Zhongjie Wang, Robert Ward, and Jia Zhang (Eds.). IEEE, 314–323. https://doi.org/10.1109/ICWS53863.2021.00049