

# Subject Classification of Software Repository

Abdelhalim Hafedh Dahou<sup>a</sup> and Brigitte Mathiak<sup>b</sup>

GESIS - Institute for Social Sciences, Germany

**Keywords:** Software Categorization, R Code, Data Augmentation, Few-Shot Learning, LLM.

**Abstract:** Software categorization involves organizing software into groups based on their behavior or domain. Traditionally, categorization has been crucial for software maintenance, aiding programmers in locating programs, identifying features, and finding similar ones within extensive code repositories. Manual categorization is expensive, tedious, and labor-intensive, leading to the growing importance of automatic categorization approaches. However, existing datasets primarily focus on technical categorization for the most common programming language, leaving a gap in other areas. This paper addresses the research problem of classifying software repositories that contain R code. The objective is to develop a classification model capable of accurately and efficiently categorizing these repositories into predefined classes with less data. The contribution of this research is twofold. Firstly, we propose a model that enables the categorization of software repositories focusing on R programming, even with a limited amount of training data. Secondly, we conduct a comprehensive empirical evaluation to assess the impact of repository features and data augmentation on automatic repository categorization. This research endeavors to advance the field of software categorization and facilitate better utilization of software repositories in the context of diverse domains research.


## 1 INTRODUCTION


Recently, the growth of open source software communities has been remarkable, leading to an exponential increase in the number of software repositories available. These repositories serve as valuable resources for developers, researchers, and users seeking to explore, reuse, and contribute to open source projects. With such a vast amount of repositories, automatic categorization and organization become crucial for efficient information retrieval and knowledge discovery and reducing manual search significantly.

Automatic software categorization involves the task of grouping software into categories that effectively describe their behavior or domain. To tackle this challenge, researchers have explored different techniques, including supervised learning, where a repository is assigned a set of categories (Sharma et al., 2017)(Thung et al., 2012). Unsupervised procedures, such as LDA, have also gained popularity as a valuable tool for grouping similar programs (Wu et al., 2016)(Tian et al., 2009). The literature has widely recognized the significance of automatic classification and the computation of repository similar-

ity.

Currently, software repositories are categorized using text classification techniques, which have been extensively studied in the field of Natural Language Processing (NLP). Impressive results have been achieved in tasks like sentiment analysis and documents topic classification. The categorization process relies on utilizing available high-level information within repositories, such as repository profiles, to accurately predict the appropriate category. However, categorizing repositories poses a significant challenge due to the distinct nature of information present in the code and accompanying textual content. Low-level terms in the source code tend to exhibit more variability compared to terms in the accompanying text. This variation arises from programmer-generated identifier names, abbreviations, compound words that are difficult to expand or split, and specific meanings assigned to words in code that differ from their English meanings (e.g., "button," "free," "object") (LeClair et al., 2018). Additionally, there is no guarantee of sufficient high-level text data being available in the repositories, which necessitates considering the source code itself rather than relying solely on high-level information. To address these challenges, a comprehensive approach that incorporates multiple dimensions of information is required

<sup>a</sup>  <https://orcid.org/0000-0001-8793-2465>

<sup>b</sup>  <https://orcid.org/0000-0003-1793-9615>

to achieve accurate and thorough classification.

In this paper, our objective is to develop a classification model capable of accurately and efficiently categorizing software repositories into predefined classes with less data taking into consideration the high-level and low-level information. The proposed model is based on a pre-trained model in order to encode the similarity between low-level terms and high-level terms in repository descriptions and code source. The contribution of this research is twofold. Firstly, we propose a model that enables the categorization of software repositories focusing on R programming, even with a limited amount of training data. Secondly, we conduct a comprehensive empirical evaluation to assess the impact of repository features level and data augmentation technique on automatic repository categorization. This research endeavors to advance the field of software categorization and facilitate better utilization of software repositories in the context of diverse domains research.

## 2 RELATED WORK

The main focus of previous research revolves around software categorization, specifically the classification of software repositories into predefined categories. Existing literature in software categorization can be divided into two categories based on the adopted methodology: machine learning (ML) and deep learning. However, the majority of studies in this domain have primarily employed ML techniques to address the problem. In recent years, an increase in the interest of software categorization due to the rise of transformer-based networks and large language models.

Starting with datasets, the (Sas and Capiluppi, 2021) presents a new dataset called LabelGit made for Github Java repositories classification based just on the source code. The dataset contains 11.502 Java projects covering 13 categories including Web, Networking, Paser, etc, resulting in around 15 GB compressed. Moving to the works that applied ML methodology, we can find (Wang et al., 2012) that aims to detect software topics by utilizing labeled descriptions through a hybrid approach that combines L-LDA and a topic detection algorithm based on a word-label matrix. The approach, called Labeled Software Topic Detection (LSTD), involves associating each project with a label, and a word-label matrix is created using L-LDA. Moreover, a topic detection algorithm is applied to the word-label matrix to identify appropriate label sets for specific projects. The effectiveness of LSTD is evaluated on a dataset

comprises project profiles with descriptions, and labels. (Linares-Vásquez et al., 2014) explores the application of ML techniques to automatically categorize open and closed Java software applications into distinct categories. The authors used API calls, including packages and classes, as attributes for categorization. Three ML algorithms (Decision Trees, Naive Bayes, and Support Vector Machines) are compared. The study examines a large dataset of Java applications from various repositories and make all case study data publicly available. The (Soll and Vosgerau, 2017) ClassifyHub is an algorithm developed for the InformatiCup 2017 competition to categorize GitHub repositories using a combination of eight weak classifiers based on ML techniques such as KNN and Decision Tree. Each classifier works on feature data, i.e., file extension, README files, commits, and repository structure. The approach has been evaluated using ten-fold cross validation over a dataset of 681 GitHub repositories with 7 classes like Solutions for homework, documents and website. (Sharma et al., 2017) exploits README files of GitHub projects to extract descriptive fragments in order to catalog them using the standard NLP techniques. The categorization has been done by using LDA-GA, a mature technique that combines Latent Dirichlet Allocation with Genetic Algorithm to build a topic model. A post-processing step was then manually conducted to remove the wrong terms or merge similar ones in terms of granularity. Their evaluation involved a dataset of 10K GitHub repositories with a minimum of 20 stars, and a user study was conducted to assign appropriate categories to each repository. Using the Github repository README file, the (Di Sipio et al., 2020) developed a tool based on a Naïve Bayesian classifier to recommend topics. The recommending process passed through: (1) Crawling dataset; (2) encoded the relevant information using the TF-IDF weight scheme. (3) Training the model and at the end add the programming language using the guessLang package. The (Di Rocco et al., 2020) presents a collaborative filtering-based recommender system named TopFilter that can suggest GitHub topics. The process of recommendation in TopFilter based on encoding repositories (README file) and related topics in a graph-based representation and applied a syntactic-based similarity function to predict missing topics from similar repositories based on TF-IDF feature vector. To enhance the prediction performance of TopFilter, they combined it with MNBN that acts as an input topic generator which leads to significant boost. The (Izadi et al., 2021) develops a Logistic Regression model using TF-IDF features for Github repository topics recommendation based

on multi-label classification technique. The data used in this study contains about 152K GitHub repositories and 228 featured topics and the learning was done on textual information.

Few works used DL techniques includes (LeClair et al., 2018) which focuses on the task of software categorization using neural network architecture. The proposed approach is evaluated using reference data from Debian end-user programs and annotated C/C++ libraries. The approach proposed used the project' function by creating vectors of integers to represent each function using the code-only and code-description representations. The neural classification model consists of CNN and LSTM layers. The voting mechanism employed is plurality voting, where the project category is determined based on the label assigned to the highest number of its functions. The proposed approach surpasses previous software classification techniques (Bag Of Words and LR) but still suffers from the vocabulary problem when the code description does not exist. In addition to that, (Zhang et al., 2019) proposes a keyword-driven hierarchical classification approach for automatically classifying GitHub repositories based on their content. It uses an HIN encoding module, a label prediction module based on MLP network, and a label refinement module to refine the predicted labels. The authors collected two datasets of GitHub repositories "Machine Learning" and "Bioinformatics". The results show that their approach outperforms several state-of-the-art methods in terms of accuracy and F1 score on both datasets.

In the realm of transformer-based and LLM, significant progress has been made, resulting in the emergence of more sophisticated methodologies. One prominent example is ChatGPT, which exhibits substantial potential across various tasks relevant to the focus of this study. These tasks encompass text classification (Kuzman et al., 2023)(Zhang et al., 2022), code comprehension and generation (Megahed et al., 2023)(Treude, 2023)(Sobania et al., 2023), data augmentation (Dai et al., 2023), and information extraction (Wei et al., 2023)(Gao et al., 2023)(Polak and Morgan, 2023). Evaluations indicate that ChatGPT's zero-shot performance is comparable to fine-tuned models such as BERT, BART, and GPT-3.5. Furthermore, leveraging advanced prompting strategies can enhance ChatGPT's comprehension capabilities. Nonetheless, it has not yet surpassed the performance of the current state-of-the-art (SOTA) models.

In contrast to the studies mentioned earlier, our work aims to categorize software repositories by utilizing both textual and source code information, providing a more comprehensive approach. Previous

research predominantly concentrated on categorizing repositories into technical classes, which lacked the ability to differentiate between repositories from various disciplines. Furthermore, these studies primarily focused on widely used programming languages like Java and Python. In contrast, our approach emphasizes categorizing repositories based on their subject or discipline, with a particular emphasis on the R programming language, which boasts a large and vibrant community. This targeted focus allows for a more specialized categorization, catering to the needs of researchers and practitioners within specific domains. Additionally, we leverage cutting-edge deep learning techniques, incorporating the vast knowledge gained from LLMs, further enhancing the accuracy and effectiveness of our categorization model.

### 3 MODEL ARCHITECTURE

Few-shot learning techniques have gained popularity as effective solutions for scenarios with limited labeled data. These techniques involve adapting pre-trained language models to specific tasks using minimal training examples. Various approaches already exist such as in-context learning (ICL) (used in GPT3), and adaptation approach which performs a localized updates that concentrate changes in a small set of model' parameters like using adapters (Houlsby et al., 2019); (Pfeiffer et al., 2020); (Üstün et al., 2020). However, they may not always be practical due to their reliance on billion-parameter language models, susceptibility to high variability caused by manually crafted prompts, and the infrastructure and cost requirements for deployment.

To address these limitations, a new approach called SETFIT (Sentence Transformer Fine Tuning) has been proposed in (Tunstall et al., 2022). SETFIT is an efficient and prompt-free approach for few-shot fine-tuning of Sentence Transformers (ST). The framework involves first fine-tuning a pretrained ST using a contrastive siamese approach with a small set of text pairs. This fine-tuned model then generates rich text embeddings, which are utilized to train a classification head (as depicted in Figure 1).

This approach has achieved impressive outcomes in text classification and offers various advantages over similar approaches like T-FEW, ADAPET, and PERFECT. It stands out for its remarkable speed during inference and training, and it doesn't rely on extensive base models or data for optimal performance, eliminating the need for external computation. Unlike previously mentioned approaches and zero-shot learning, SETFIT avoids the instabil-

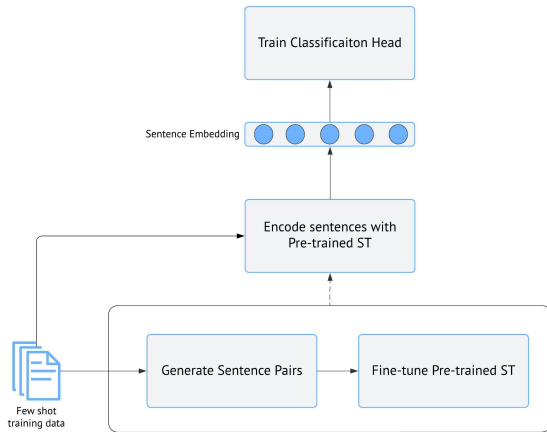


Figure 1: A visual representation of *SetFIT*'s architecture.

ity and inconvenience associated with prompting. Considering our case of limited data and real-world deployment, this approach is highly suitable.

**ST Fine-Tuning.** To overcome the limitations posed by the scarcity of data, the initial phase of fine-tuning the ST model relies on a contrastive training approach. This technique, commonly employed for image similarity (Koch et al., 2015), involves the generation of positive and negative triplets from a small labeled example set,  $D = (x_i, y_i)$ , where  $x_i$  represents sentences and  $y_i$  represents their corresponding class labels. For each class label  $c \in C$ , two sets of triplets are formed:  $R$  positive triplets ( $T_{c_p}$ ) comprising pairs from the same class, and  $R$  negative triplets ( $T_{c_n}$ ) containing a sentence from one class and another from a different class. During each iteration of sentence-pair generation,  $2 \times K$  training pairs are created, with  $K$  representing the total number of training samples in the task. The contrastive fine-tuning dataset, denoted as  $T$ , is constructed by concatenating the positive and negative triplets across all class labels. Subsequently, the sentence transformer model undergoes fine-tuning on the  $T$  dataset to yield an adapted ST.

**Classification Head and Inference.** In the second phase, the limited original training data is employed to generate sentence embeddings using the adapted ST. These embeddings, along with the original class labels, are used as the dataset to train the classification head. In the original research paper, a logistic regression (LR) model is utilized as the text classification head. During the inference phase, each test sentence undergoes encoding using the adapted ST, and the LR model predicts its corresponding category.

## 4 EXPERIMENTAL DESIGN

### 4.1 Dataset

Due to the lack of an existing dataset specialized in classifying projects written in R language to predefined disciplines, we took the initiative to collect our own dataset from the Zenodo<sup>1</sup> platform. Zenodo is an open repository, serving as a versatile platform. It enables researchers to upload various forms of digital artifacts related to their research, including research papers, datasets, research software, reports, and other research-related materials. Our dataset comprises a total of 479 projects, encompassing various classes such as 82 Bioinformatics, 97 Economics, 100 Social Science, 100 Statistical and Data analysis, and 100 Environmental Science. In order to capture comprehensive information about each project, we collected multiple features, including the title, description, content (code), and file names. Additionally, we generated additional features such as keywords and code descriptions using the GPT-4 via API calls, to provide a more in-depth understanding of the projects within our dataset. We divided the dataset into train and test parts by using a balanced distribution between the classes which resulted in 250 projects (50 projects in each class) for training and rest for testing.

### 4.2 ST Model Selection

When it comes to selecting the optimal model for a specific downstream task, explicit guidelines are often absent. Using the hypothesis mentioned in (Tunstall et al., 2022) by employing an embedding model trained to identify semantic similarity between sentence pairs due to similarity between classes in a text classification setup. Consequently, we focused on embedding models trained with all or paraphrase datasets. Our selection process concentrated on models that demonstrated superior sentence embedding performance and have a small size in terms of parameters as seen in Table 1.

Table 1: Candidate ST models that were chosen for validating SetFit on the our dataset.

Model	Size	Input
all-MiniLM-L6-v2	80 MB	256
paraphrase-MiniLM-L6-v2	80 MB	384
paraphrase-mpnet-base-v2	970 MB	512
all-mpnet-base-v2	420 MB	768

<sup>1</sup><https://zenodo.org/>



### 4.3 Baselines

Most of the previous works in software categorization applied ML and DL techniques which required a large dataset. Due to the non existing data in software disciplines categorization, we compare the few shot approach performance using our small dataset against recent best LLM performing zero-shot approaches such as: GPT (Generative Pre-trained Transformer) series including CharGPT and GPT-4 (OpenAI, 2023), BART model (Lewis et al., 2019) and CodeBERT model (Feng et al., 2020).

**GPT3.5 and GPT-4** are the latest iterations of OpenAI’s GPT model, representing advanced versions developed for understanding and generating both natural language and code. These models are pre-trained on extensive text data, including natural language and programming language. Although specific details about their training data and architecture are not well-documented, they are believed to be autoregressive language models based on the transformer architecture (Vaswani et al., 2017). In our experiments, we utilize the *gpt-3.5-turbo* and *gpt-4* models through the official API, without making any modifications to the default parameters.

**BART** is a transformer-based encoder-decoder model, utilizing both bidirectional encoding similar to BERT and autoregressive decoding akin to GPT. BART is pre-trained through a two-step process: (1) introducing noise to the text using an arbitrary function, and (2) training the model to reconstruct the original text. The architecture consists of 12 layers each in the encoder and decoder components. In this study, we used the *bart-large-mnli*<sup>2</sup> which is a bart base model that was fine-tuned using the MultiNLI (MNLI) dataset (Williams et al., 2017), enabling its utilization for zero-shot text classification tasks.

**CodeBERT** is a pre-trained model designed to handle both programming and natural languages. It is trained on a combination of textual and code data from CodeSearchNet (Husain et al., 2019), employing the MLM+RTD (Masked Language Modeling + Replaced Token Detection) objectives. CodeBERT is developed using the identical model architecture as RoBERTa-base, as described in the work by Liu et al. (2019). It consists of a total of 125 million model parameters.

<sup>2</sup><https://huggingface.co/facebook/bart-large-mnli>

### 4.4 Experimental Setup

In the few-shot framework, we conducted fine-tuning of the ST model by utilizing a cosine-similarity loss. The fine-tuning process involved a learning rate of 1e-3, a batch size of 16, and a maximum sequence length of 256 tokens. We performed the fine-tuning for 5 epochs by fixing the number of iteration to 5. In all our experiments, we made use of the Hugging Face Transformers library (Wolf et al., 2020).

To generate features using the GPT-4 model, we employed the prompt *“Generate keywords from this repository description and code:”* including the title, description, and code.

For the zero-shot classification, we used the following prompt structure: *“Classify this repository information into one of the following classes: [Bioinformatics, Environmental Science, Social Sciences, Statistics and Data Analysis, Economics]”* with different combinations of features.

## 5 RESULTS AND DISCUSSION

This section presents the evaluation of our approach in comparison to the baselines, with the objective of addressing various research questions that pertain to different aspects of input features and classifier models.

- RQ1. How does the performance of the model differ when using only text descriptions compared to using text descriptions along with source code?
- RQ2. How does generating new features (transforming the source code into a text description and generating repository’ keywords) affect the performance of the models?
- RQ3. Does combining different features actually enhance the accuracy of the models?
- RQ4. Is it possible for data augmentation to enhance the accuracy of our approach?

### 5.1 RQ1. Performance on Text Description and Code

This section presents the evaluation results for the models based on their performance using text descriptions alone (table 2) and text descriptions with code sources (table 3). In the following, we will discuss the results and answer the first research question.

Upon comparing the two tables, it becomes apparent that the performance of the models deteriorates when code sources are incorporated alongside

Table 2: Evaluation results based on text description feature.

Models	Acc.	Pre.	Rec.	F1
Bart-large-mnli	0.49	0.57	0.5	0.47
GPT3.5	0.35	0.52	0.37	0.32
GPT-4	0.38	0.48	0.39	0.35
CodeBERT	0.26	0.34	0.24	0.17
Few-shot <sub>all</sub> -MiniLM	0.57	0.59	0.55	0.56

Table 3: Text description with code source evaluation.

Models	Acc.	Pre.	Rec.	F1
Bart-large-mnli	0.27	0.31	0.29	0.25
GPT3.5	0.29	0.41	0.32	0.26
GPT-4	0.32	0.31	0.34	0.26
CodeBERT	0.11	0.1	0.12	0.09
Few-shot <sub>all</sub> -MiniLM	0.47	0.52	0.51	0.48

text descriptions (table 3). This decline in performance is consistent across all evaluated models, as indicated by the decreased accuracy, precision, recall, and F1 scores. This outcome is not entirely surprising, considering that the algorithms were primarily developed for achieving high performance on text data rather than low-level data. From our perspective, these results suggest that classifying source code presents a distinct challenge compared to text classification, likely due to the vocabulary disparities.

CodeBERT, in particular, demonstrates the most notable decline in performance, showcasing the least favorable results compared to all other models in table 3. Despite being trained on both text descriptions and code sources, CodeBERT continues to face difficulties. One possible explanation for this could be the distinct structural differences between the R code used and the programming language encountered during the training phase. Additionally, CodeBERT may have a relatively limited understanding of textual information compared to the other models, which could contribute to its struggles in this task.

Few-shot approach manages to maintain relatively high performance in both experiments. This indicates that it is more robust in handling the combined input types and demonstrates a better understanding of the relationship between text descriptions and code sources even with few samples. GPT3.5's accuracy decreases from 0.35 to 0.29, indicating a reduction in its ability to correctly classify repositories. Similarly, GPT-4's accuracy drops from 0.38 to 0.32 which suggests that the inclusion of code sources in the input negatively affects the performance of both GPT3.5 and GPT-4 which confirms that classifying source code is a different problem than text classification even when the input length touches the 8k tokens.

## 5.2 RQ2. New Features Generating

The provided section presents the evaluation results (Table 4) of all models based on their performance when transforming code sources into text descriptions (Explanation of the code) and generating of keywords using the GPT-4 model.

Table 4: Transforming code source to text description evaluation.

Models	Acc.	Pre.	Rec.	F1
Bart-large-mnli	0.29	0.48	0.29	0.21
GPT3.5	0.28	0.38	0.31	0.25
GPT-4	0.28	0.42	0.31	0.26
CodeBERT	0.22	0.16	0.22	0.16
Few-shot <sub>all</sub> -MiniLM	0.33	0.36	0.34	0.34

Looking at the results in table 4, it is evident that transforming code sources into text descriptions negatively impacts the performance of the evaluated models. The metrics scores are generally lower compared to the previous tables. Among the models, Few-shot approach exhibits the highest performance with an accuracy of 0.33 and an F1 score of 0.34. Bart-large-mnli, GPT3.5, and GPT-4 show similar performances, with comparable accuracy and F1 scores. However, their precision and recall vary slightly. The drop may be caused by the misunderstanding of the GPT3.5 of the code source even with adding the title and description to the prompt in order to give more information about the repository.

Table 5: Title, description and keywords evaluation.

Models	Acc.	Pre.	Rec.	F1
Bart-large-mnli	0.49	0.58	0.5	0.48
GPT3.5	0.31	0.37	0.33	0.26
GPT-4	0.32	0.43	0.33	0.27
CodeBERT	0.24	0.13	0.22	0.14
Few-shot <sub>all</sub> -MiniLM	0.62	0.7	0.62	0.64

The results in table 5 indicate variations in the performance of the models when considering title, description, and keywords. Few-shot approach demonstrates the highest performance among the models, with the highest scores over all metrics. This indicates that Few-shot is more effective at utilizing the title, description, and keywords to classify correctly the repositories. Moreover, Bart-large-mnli performs relatively well, achieving a moderate accuracy of 0.49 and a reasonably high precision of 0.58. However, its recall and F1 score are slightly lower, suggesting that it may struggle with capturing all relevant information from the title, description, and keywords. GPT3.5 and GPT-4 show similar performances with

lower scores compared to Bart-large-mnli and Few-shot. CodeBERT exhibits the lowest performance among the models and still struggles with the classification of the repositories.

### 5.3 RQ3. Combining Features

The results in table 6 demonstrate the performance of the models when combining all the repository features including: title, description, keywords, code description, and file names.

Table 6: Evaluation results based on all input features.

Models	Acc.	Pre.	Rec.	F1
Bart-large-mnli	0.34	0.54	0.35	0.29
GPT3.5	0.35	0.51	0.38	0.32
GPT-4	0.35	0.57	0.37	0.31
CodeBERT	0.21	0.33	0.19	0.13
Few-shot <sub>all</sub> -MiniLM	0.51	0.6	0.52	0.52

The Few-shot approach performs the best among the models, achieving the highest results. Bart-large-mnli, GPT3.5, and GPT-4 exhibit similar performances, with comparable accuracy. While their performance is better than CodeBERT, they still show relatively lower scores compared to the Few-shot. The introduction of code description and file name in the table 6 resulted in a decrease in performance. This can be attributed to the inclusion of confusing information that impacted the classifier, particularly due to the overlap between the five classes. The file names mostly had a connection to statistical and data analysis, introducing a bias in the results. Similarly, the code descriptions generated by GPT3.5 followed a structure related to the statistical and data analysis class, using technical terms like distribution, modeling, programming, etc. Therefore, to achieve high performance, especially when employing deep learning approaches, it is crucial to incorporate a feature selection phase by comparing the behavior of the classifier over each feature and combination of features. By doing so, we can optimize the model's performance and mitigate biases introduced by certain features.

### 5.4 RQ4. Data Augmentation Impact

Table 7 displays the evaluation outcomes concerning the influence of data augmentation on the Few-shot approach. The augmentation was performed using the GPT3.5 model to produce a rephrased version of the title, description, and keywords. The selection of these features was based on the excellent performance exhibited by the approach when solely utilizing the ti-

tle, description, and keywords features.

Table 7: Data Augmentation (Aug.) impact evaluation for Few-shot model.

Models	Acc.	Pre.	Rec	F1
Without Aug.	0.62	0.7	0.62	0.64
Aug.	0.56	0.7	0.54	0.55

Comparing the two scenarios, we observe that Few-shot without augmentation achieves higher results compared to Few-shot with augmentation. This indicates that data augmentation has a negative impact on the model's performance in this specific case. The decrease in performance can be attributed to several factors: the augmented data (i) may introduce noise or irrelevant patterns, (ii) not adequately capture the underlying patterns and characteristics of the input, and (iii) may introduce biases or inconsistencies that negatively affect the model's ability to generalize to new inputs.

## 6 CONCLUSIONS

In conclusion, the study highlights the potential of the few-shot learning approach to achieve promising results even with limited labeled data. The performance of all approaches examined in this research was significantly impacted by the presence of high-level text data. However, the challenge of overlapping categories remains a persistent issue in the task of software repository classification. The diversity in code sources introduces additional ambiguity due to variations in vocabulary usage. Furthermore, fine-tuning a pre-trained model specifically for this task demonstrated the capability to surpass the capabilities of LLMs. Leveraging LLMs, however, proved valuable in generating more helpful features for the classification process. These findings emphasize the importance of incorporating both advanced techniques like few-shot learning and utilizing the strengths of LLMs to enhance software categorization tasks. Further research and development in this field hold the potential to refine and optimize the automatic classification of software repositories by evaluating more approaches, use more input features and enrich the source code with line comments.

## REFERENCES

Dai, H., Liu, Z., Liao, W., Huang, X., Wu, Z., Zhao, L., Liu, W., Liu, N., Li, S., Zhu, D., et al. (2023). Chataug:

- Leveraging chatgpt for text data augmentation. *arXiv preprint arXiv:2302.13007*.
- Di Rocco, J., Di Ruscio, D., Di Sipio, C., Nguyen, P., and Rubei, R. (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)*, pages 1–11.
- Di Sipio, C., Rubei, R., Di Ruscio, D., and Nguyen, P. T. (2020). A multinomial naïve bayesian (mnb) network to automatically recommend topics for github repositories. In *Proceedings of the Evaluation and Assessment in Software Engineering*, pages 71–80.
- Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al. (2020). Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.
- Gao, J., Zhao, H., Yu, C., and Xu, R. (2023). Exploring the feasibility of chatgpt for event extraction. *arXiv preprint arXiv:2303.03836*.
- Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.
- Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. (2019). Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.
- Izadi, M., Heydarnoori, A., and Gousios, G. (2021). Topic recommendation for software repositories using multi-label classification algorithms. *Empirical Software Engineering*, 26:1–33.
- Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille.
- Kuzman, T., Mozetic, I., and Ljubešić, N. (2023). Chatgpt: Beginning of an end of manual linguistic data annotation? use case of automatic genre identification. *ArXiv, abs/2303.03953*.
- LeClair, A., Eberhart, Z., and McMillan, C. (2018). Adapting neural text classification for improved software categorization. In *2018 IEEE international conference on software maintenance and evolution (IC-SME)*, pages 461–472. IEEE.
- Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.
- Linares-Vásquez, M., McMillan, C., Poshyvanyk, D., and Grechanik, M. (2014). On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering*, 19:582–618.
- Megahed, F. M., Chen, Y.-J., Ferris, J. A., Knuth, S., and Jones-Farmer, L. A. (2023). How generative ai models such as chatgpt can be (mis) used in spc practice, education, and research? an exploratory study. *Quality Engineering*, pages 1–29.
- OpenAI (2023). Gpt-4 technical report. *arXiv*.
- Pfeiffer, J., Vulić, I., Gurevych, I., and Ruder, S. (2020). Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*.
- Polak, M. P. and Morgan, D. (2023). Extracting accurate materials data from research papers with conversational language models and prompt engineering—example of chatgpt. *arXiv preprint arXiv:2303.05352*.
- Sas, C. and Capiluppi, A. (2021). Labelgit: A dataset for software repositories classification using attributed dependency graphs. *arXiv preprint arXiv:2103.08890*.
- Sharma, A., Thung, F., Kochhar, P. S., Sulistya, A., and Lo, D. (2017). Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 314–319.
- Sobania, D., Briesch, M., Hanna, C., and Petke, J. (2023). An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653*.
- Soll, M. and Vosgerau, M. (2017). Classifyhub: an algorithm to classify github repositories. In *KI 2017: Advances in Artificial Intelligence: 40th Annual German Conference on AI, Dortmund, Germany, September 25–29, 2017, Proceedings 40*, pages 373–379. Springer.
- Thung, F., Lo, D., and Jiang, L. (2012). Detecting similar applications with collaborative tagging. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 600–603. IEEE.
- Tian, K., Revelle, M., and Poshyvanyk, D. (2009). Using latent dirichlet allocation for automatic categorization of software. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 163–166. IEEE.
- Treude, C. (2023). Navigating complexity in software engineering: A prototype for comparing gpt-n solutions. *arXiv preprint arXiv:2301.12169*.
- Tunstall, L., Reimers, N., Jo, U. E. S., Bates, L., Korat, D., Wasserblat, M., and Pereg, O. (2022). Efficient few-shot learning without prompts. *arXiv preprint arXiv:2209.11055*.
- Üstün, A., Bisazza, A., Bouma, G., and van Noord, G. (2020). Uadapter: Language adaptation for truly universal dependency parsing. *arXiv preprint arXiv:2004.14327*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Wang, T., Yin, G., Li, X., and Wang, H. (2012). Labeled topic detection of open source software from mining mass textual project profiles. In *Proceedings of the First International Workshop on Software Mining*, pages 17–24.
- Wei, X., Cui, X., Cheng, N., Wang, X., Zhang, X., Huang, S., Xie, P., Xu, J., Chen, Y., Zhang, M., et al. (2023).



- Zero-shot information extraction via chatting with chatgpt. *arXiv preprint arXiv:2302.10205*.
- Williams, A., Nangia, N., and Bowman, S. R. (2017). A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.
- Wu, Y., Yao, Y., Xu, F., Tong, H., and Lu, J. (2016). Tag2word: Using tags to generate words for content based tag recommendation. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 2287–2292.
- Zhang, B., Ding, D., and Jing, L. (2022). How would stance detection techniques evolve after the launch of chatgpt? *arXiv preprint arXiv:2212.14548*.
- Zhang, Y., Xu, F. F., Li, S., Meng, Y., Wang, X., Li, Q., and Han, J. (2019). Higitclass: Keyword-driven hierarchical classification of github repositories. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 876–885. IEEE.

