Automatic Generation of Comparison Charts of Similar GitHub Repositories from Readme Files

Emad Albassam
Department of Computer Science-Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia

Abstract—**GitHub is a widely used platform for hosting open-source projects, with over 420 million repositories, promoting code sharing and reusability. However, with this tremendous number of repositories, finding a desirable repository based on user needs takes time and effort, especially as the number of candidate repositories increases. A user search can result in thousands of matching results, whereas GitHub shows only basic information about each repository. Therefore, users evaluate repositories' applicability to their needs by inspecting the documentation of each repository. This paper discusses how comparison charts of similar repositories can be automatically generated to assist users in finding the desirable repository, reducing the time required to inspect their readme files. First, we implement an unsupervised, keyword-driven classifier based on the Lbl2TransformerVec algorithm to classify relevant content of GitHub readme files. The classifier was trained on a dataset of readme files collected from Java, JavaScript, C#, and C++ repositories. The classifier is evaluated against a different dataset of readme files obtained from Python repositories. Evaluation results indicate an F1 score of 0.75. Then, we incorporate rule-based adjustments to enhance classification results by 13%. Finally, the unique features, similarities, and limitations are automatically extracted from readme files to generate comparison charts using Large Language Models (LLMs).**

Keywords*—*Multi-class classification; keyword-driven classification; rule-based classification; unsupervised classification; GitHub repositories; comparison charts

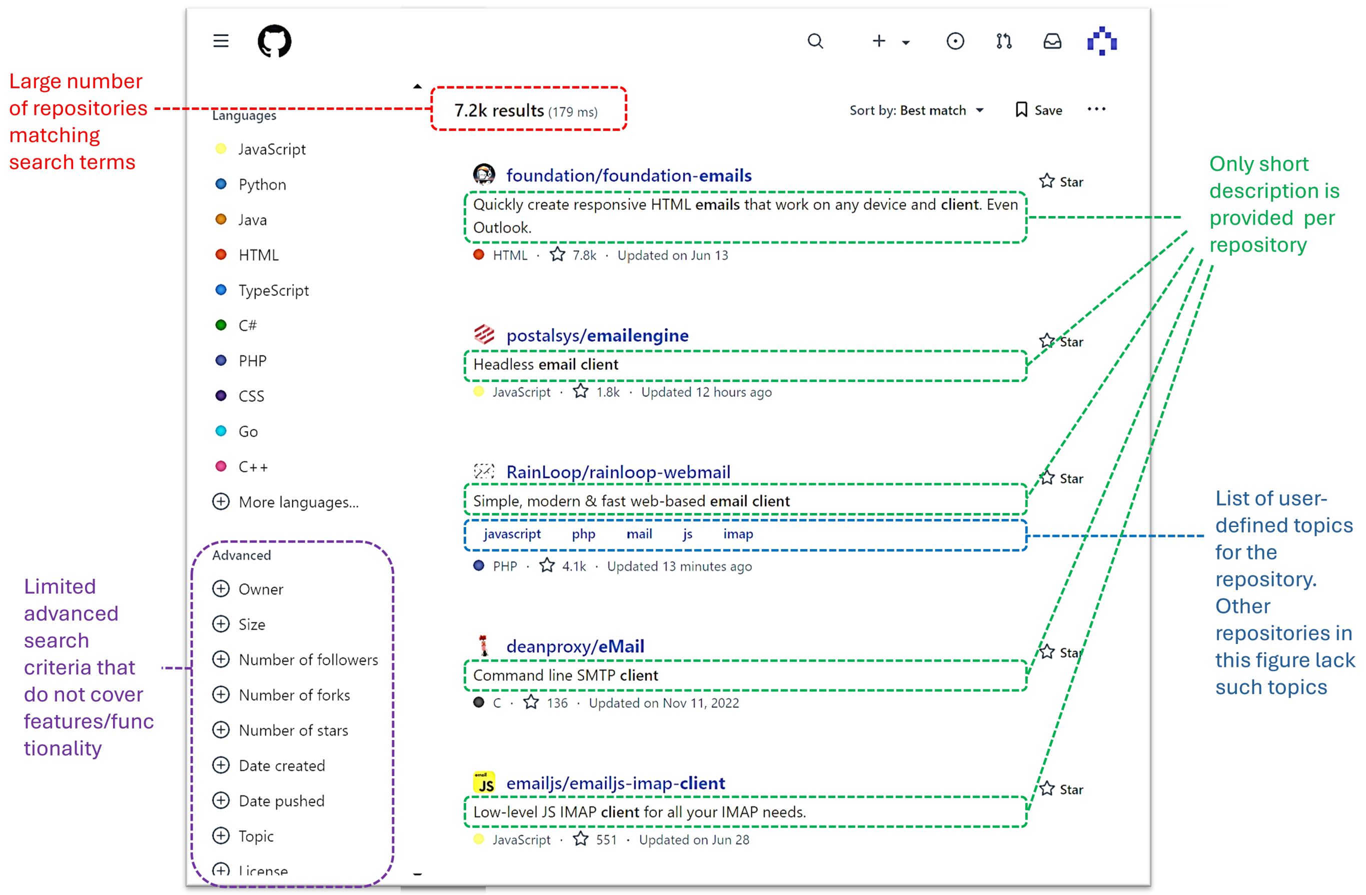
I. Introduction

Publicly available code repositories are widely used for managing and sharing application source codes. Due to their publicity, external users can review and improve these repositories, increasing their quality. Therefore, software developers engaged in development projects rely on these repositories to achieve high reusability. One such widely used code repository platform is GitHub. A recent report shows that GitHub hosts 420 million repositories with over 100 million registered developers [1]. However, this tremendous number of repositories introduces the challenge of finding a repository that adequately satisfies the user's needs.

In recent years, there has been considerable interest in cataloging the growing number of public repositories to facilitate the search, identification, and selection of these repositories for end-users. Several supervised approaches have been investigated in which the available textual documentation of public repositories is collected, analyzed, and classified to address this problem (e.g. [2] [3]). However, such supervised approaches require human intervention to manually label large datasets for training. In contrast, unsupervised approaches do not require such manual efforts since they can learn hidden patterns from large datasets. Therefore, labeled data are not required. Although several unsupervised approaches for cataloging public repositories have been proposed (e.g. [4]), they focus on the problem of tagging them with a limited set of topics. Therefore, these topics do not represent all of the capabilities provided by the corresponding repositories. As a result, users often need to read the documentation of each repository to understand its capability. This makes manual searching for candidate repositories complex and time-consuming since users need to inspect and read the documentation associated with each repository to understand their advantages and limitations before deciding. We consider the case in which a user knows the type of repository they are looking for but not the exact, full features provided by the repository. Furthermore, without careful inspection of their documentation, users might neglect important features during repository selection, which might be decided to be necessary after adopting another repository lacking these features.

As a motivation example, Fig. 1 shows typical repository search results performed on GitHub, where the user search terms include "email client", and results are sorted based on best matches. This figure shows that the search results by GitHub include approximately 7.1k repositories that match the search terms. Furthermore, only basic information is displayed for each repository. The results include a list of topics for some repositories. However, repository owners set these topics manually, which can be incomplete, error-prone, or missing in many repositories [4]. Therefore, users need to inspect each repository individually by reading its documentation to assess its relevance to their needs. In addition, users might be aware only of a subset of the features and functionality they desire, neglecting other features that could be equally important during selection.

Concerning these challenges, this work contributes to the literature by discussing how comparison charts of similar GitHub repositories can be automatically generated from their readme files using unsupervised learning. Each comparison chart consists of a set of N user-selected repositories listing (1) each repository's provided features and functionality, (2) the commonality between these repositories, and (3) the limitations of each repository, thus minimizing the effort and time to inspect and compare these repositories individually. Second, we show how the predictive performance of the Lbl2TransformerVec algorithm [5] [6] [7], an algorithm for unsupervised document classification and retrieval, can be improved by rule-based adjustments to classify the content of readme files in cases where multiple classes have approximately similar scores.


Large number of repositories matching search terms
Only short description is provided per repository
List of user-defined topics for the repository. Other repositories in this figure lack such topics
Limited advanced search criteria that do not cover features/functionality
Languages
JavaScript
Python
Java
HTML
TypeScript
C#
PHP
CSS
Go
C++
More languages...
Advanced
Owner
Size
Number of followers
Number of forks
Number of stars
Date created
Date pushed
Topic
License
Sort by: Best match
Save
Star

7.2k results (179 ms)

foundation/foundation-emails
Quickly create responsive HTML emails that work on any device and client. Even Outlook.
HTML · 7.8k · Updated on Jun 13

postalsys/emailengine
Headless email client
JavaScript · 1.8k · Updated 12 hours ago

RainLoop/rainloop-webmail
Simple, modern & fast web-based email client
javascript php mail js imap
PHP · 4.1k · Updated 13 minutes ago

deanproxy/eMail
Command line SMTP client
C · 136 · Updated on Nov 11, 2022

emailjs/emailjs-imap-client
Low-level JS IMAP client for all your IMAP needs.
JavaScript · 551 · Updated on Jun 28

Fig. 1. An example of GitHub search results, with over 7.1k results satisfying user search terms.

The remainder of this paper is organized as follows. Section II contains the related works. Section III describes the architecture of the proposed. This includes data collection, preprocessing, and model training. Section IV presents our results of evaluating the various models incorporated to generate comparison charts. Section V discusses the strengths and limitations of the proposed solution and highlights future research directions. Section VI concludes our work.

II. Related Works

Many prior studies investigated the problem of analyzing GitHub repositories from their readme files. Work by Prana et al. [2] showed how a multi-label classifier can categorize GitHub readme files. Their approach labels the content based on eight categories: what, why, how, when, who, references, contribution, and others. The approach incorporates supervised learning where readme files obtained from GitHub repositories are manually analyzed and labeled. The work by Wu et al. [3] shows how repositories can be retrieved through functional semantics where readme files need to be manually inspected. Their dataset is based on JavaScript repositories. However, their data preprocessing involves the removal of many contents that are outside of functionality, including how to use the repository. Compared to these approaches, our work considers an unsupervised approach in which the training dataset is not labeled. Furthermore, we focus here on the comparison of different GitHub repositories.

Prior works have investigated different types of tasks related to GitHub repositories. Work by Zhou et al. [8] proposed an approach for recommending GitHub trending repositories. Sipio et al. [9] investigated a topic recommendation system of GitHub repositories, which repository developers can use to label their repositories correctly. However, their work uses a supervised model. Prior works have also considered categorizing GitHub repositories based on functionality [10] and application domains [11]. Compared to these works, we focus on comparing different repositories in a single artifact when users do not have the full knowledge of the features they seek in repositories. Although it is possible to tag software repositories from their bytecode and dependencies among them [12], such approaches are considered technology-specific. We consider in this work an approach that relies on textual readme files, which makes it applicable to any repositories with such files [13]. The work of Zhang et al. [4] presented a keyword-driven hierarchical classification of GitHub repositories to assign topic labels to GitHub repositories. Their work is unsupervised but requires users to provide one keyword for each class. Although topics play a significant role in the cataloging of repositories, this work considers a different problem where similar repositories need to be compared.

Several prior works investigated the nature and quality of documentation available in GitHub repositories. Results of Liu et al. [14] show that readme files of open-source Java projects do not align with GitHub guidelines. Furthermore, work by

Venigalla and Chimalakonda [15] shows that the presence of readme files, lists, images, and links increases the popularity of repositories. The work by Treude et al. [16] provides an assessment of documentation quality in ten dimensions, where their results show that documentation of various artifacts such as references, documents, and articles are different in terms of quality. Elazhary et al. [17] investigated GitHub developer contribution guidelines through a mixed-method study of 53 GitHub projects, where their results show that approximately 68% of these projects diverge significantly from the expected process model. Venigalla and Chimalakonda [18] investigated software documentation on GitHub and showed that multiple software artifacts can contribute to documentation. Work by Hellman et al. [19] proposed an approach for generating GitHub repository descriptions. Their analysis showed that descriptions of GitHub repositories are poor due to a lack of purpose in their description. These works clearly show the challenges associated with analyzing the documentation of publicly available repositories, such as lack of standardization and quality immaturity. This paper investigates several such challenges to generating meaningful comparison charts to end users.

Table I summarizes the research limitations identified in prior works. While all prior works focus on individual repositories to generate topics, the proposed approach considers the generation of comparison charts of multiple repositories. This approach is not limited to identifying the features provided by repositories but also identifies their commonalities and limitations, which, to the best of our knowledge, have not been investigated previously in prior works.

III. Automatic Generation of Comparison Charts

This section first provides an overview of the proposed solution for generating comparison charts from GitHub readme files. Then, we discuss each process within the architecture, including data collection, data preprocessing, and model training processes, where the goal is to implement a multi-class classifier capable of classifying the sections of a given GitHub readme file that are likely to contain the features and functionalities provided by the corresponding repository. We then discuss each process related to the automatic generation of comparison charts.

A. Overview of Proposed Solution

The architecture of the proposed solution (see Fig. 2) consists of two modules: offline and online. The offline module is performed once to train a multi-class classifier capable of classifying the contents of readme files obtained from GitHub repositories. The model is keyword-driven and is trained to classify the various sections based on the likelihood of containing relevant information for constructing comparison charts. The model is complemented with rule-based heuristics to adjust the classifier's results when multiple classes have approximately close scores by the classifier.

On the other hand, the online module is responsible for generating comparison charts of GitHub repositories that satisfy the user search terms. First, this module performs a live search of GitHub repositories using user-provided search terms. The user then selects *N* repositories from the search results they wish to compare. The online module then automatically retrieves the readme files corresponding to the user-selected *N* repositories and incorporates the hybrid classifier from the offline module to extract relevant sections required by subsequent processes. The online module then extracts relevant information, including provided features and limitations, from these sections and computes the similarity of features from the different repositories to generate comparison charts.

We describe each process in the architecture in the following subsections.

TABLE I. Limitations of Literature

Ref.	Limitations
[2]	• Supervised learning approach requiring manual labeling of the training set. • Proposed approach does not consider comparison of different repositories and does not extract unsupported features/functionalities of repositories.
[3]	• Manual inspection of GitHub readme files. • Manual removal of all sections in readme files except sections related to functionality. • Proposed approach does not consider comparison of different repositories and does not extract unsupported features/functionalities of repositories. • Dataset is limited to Javascript repositories.
[4]	• Proposed approach requires user providing one keyword for each class as guidance (Although a keyword enrichment process module is incorporated to expand the single keyword to a keyword set for each category). • Proposed approach does not consider comparison of different repositories and does not extract unsupported features/functionalities of repositories. • Dataset is limited to Machine Learning and Bioinformatics domains.
[13]	• Featured topics may neglect detailed functionalities provided by repositories. Thus, users still need to inspect individual readme files to understand their capabilities. • Featured topics may evolve, which may require retraining of the supervised models. • Proposed approach does not consider comparison of different repositories and does not extract unsupported features/functionalities of repositories.
[10]	• Proposed approach is semi-automated, with steps requiring manual human intervention. • Functionalities are extracted from readme file segments by computing the similarity between the segments and a short 1-2 lines of description from the repository's homepage, which may result in missed functionalities. • Proposed approach does not consider comparison of different repositories and does not extract unsupported features/functionalities of repositories.
[20]	• Proposed approach recommends trending repositories for developers based on their historical commits on GitHub (i.e. does not target general users of GitHub). • Proposed approach does not consider comparison of different repositories and does not extract unsupported features/functionalities of repositories.

B. Automatic Data Collection

To prepare the dataset used in this work, we collected 283 readme files obtained from GitHub repositories. These

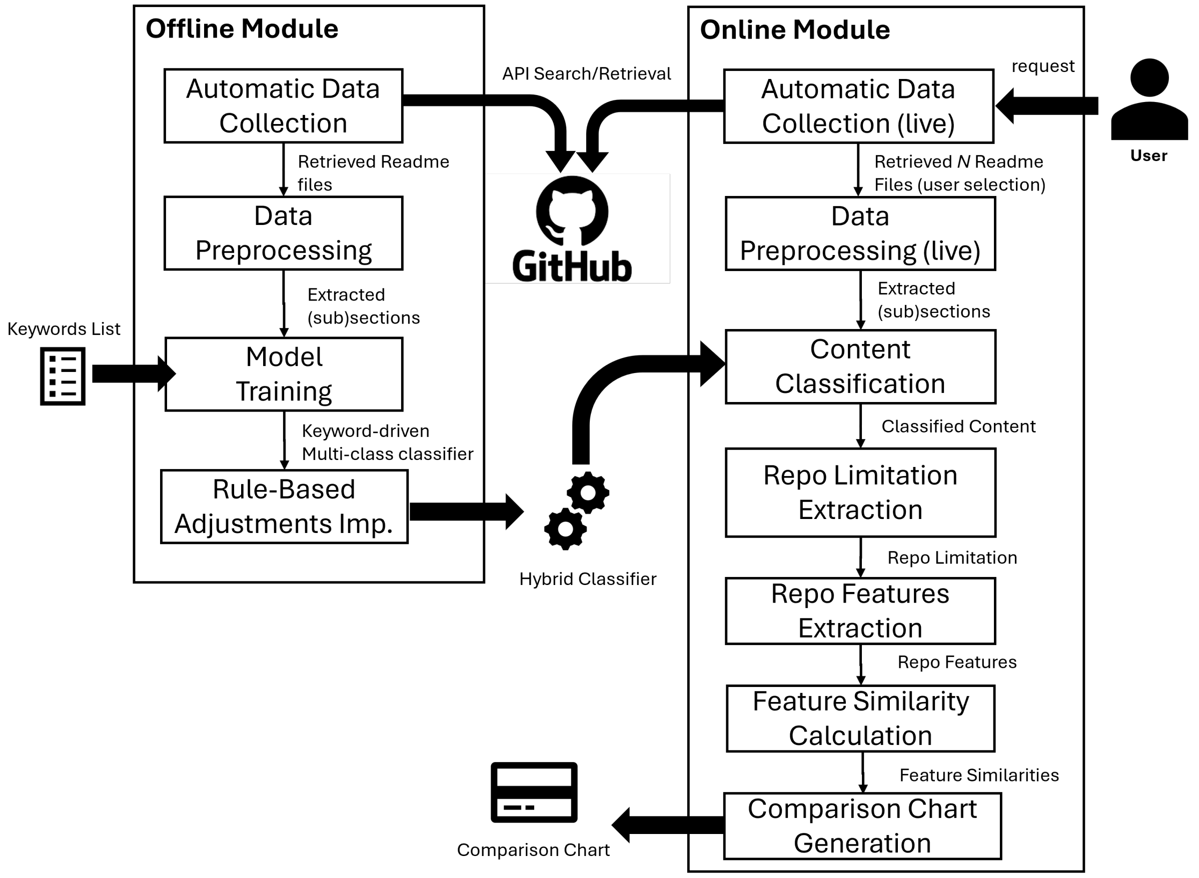


Fig. 2. An overview of the architecture for generating comparison charts.

files are collected automatically through GitHub's Representational State Transfer (REST) API for searching [21]. To ensure diverse coverage of repositories, we retrieve repositories whose primary programming language is Java, JavaScript, C#, Python, or C++. We obtain the readme files of the top 100 repositories for each of these languages, where the results are sorted in descending order based on repository stars. Then, we filter the results by including only the repositories with readme files written in English and exceeding 1000 bytes. These filtering rules aim to include only mature repositories with high stars, which increases the likelihood of obtaining well-written readme files.

We further split the collected readme files into two classes: training and testing. The training class contains the readme files corresponding to repositories whose primary programming languages are Java, JavaScript, C#, and C++. On the other hand, the readme files related to Python will be used for testing purposes to evaluate the hybrid classifier. By splitting the readme files based on the programming language, we decrease the likelihood of language bias during evaluation since our model is never trained on readme files related to the repositories related to the Python language. Table II summarizes the number of readme files collected from GitHub and shows the percentages of training and testing classes. As seen in this table, selecting the readme files related to the Python language as the testing class splits the collected readme files at an approximately 80:20 ratio.

C. Data Preprocessing

The contents of collected readme files are cleansed as follows. Embedded HTML tags and elements (such as HTML comments) are removed. All code blocks and URLs that appear in readme files are replaced with the constant strings *@code* and *@link*, respectively. Irrelevant content, such as images, task lists, color codes, and emojis, are removed.

After data cleansing, we extract the heading and content of all sections and subsections found in the readme files using regular expressions, which, according to GitHub formatting syntax [22], start with '#' symbols. Each extracted (sub)section represents an instance in our dataset. For each instance, we also record (1) the GitHub repository ID to maintain traceability between the instances and the files from which they were extracted, (2) the level of the (sub)section heading, which can range from 1 (i.e. a first-level heading) to 6 (a sixth-level heading), and (3) the order of the (sub)section in the document. Table III summarizes the number of instances obtained after preprocessing and instance extraction grouped by programming language. As can be seen, the training set accounts for 78.2% of the number of instances, while the testing set accounts for 21.5%.

D. Keyword-Based Model Training

After data preprocessing, we trained various keyword-driven models based on the Lbl2TransformerVec algorithm,

TABLE II. Summary of Data Collection Process of Readme Files Obtained from GitHub Repositories

Language	Total Number of Readme Files			Class	Pct.
	Initial	After Excl. Non-English	After Excl. files $<$ 1KB		
Java	100 files	45 files (55 files excl.)	43 files (2 files excl.)	Training	12.7%
JavaScript	100 files	73 files (27 files excl.)	71 files (2 files excl.)	Training	20.9%
C#	100 files	83 files (17 files excl.)	80 files (3 files excl.)	Training	23.6%
C++	100 files	81 files (19 files excl.)	78 files (3 files excl.)	Training	23%
Python	100 files	67 files (33 files excl.)	67 files (0 files excl.)	Testing	19.8%

TABLE III. Summary of Extracted Instances from Collected Readme Files

Language	No. of Instances	Class	Pct.
Java	617	Training	13.5%
JavaScript	1112	Training	24.35%
C#	977	Training	21.4%
C++	876	Training	19.2%
Python	984	Testing	21.55%

an algorithm for unsupervised document classification and retrieval that does not require stemming or lemmatization and can work on short texts [5] [6] [7]. For each model we train, we incorporate a different transformer, as shown in Table IV, where the goal is to evaluate the impact of the various transformers on the classifier's classification results. The models are trained on the training set corresponding to the Java, JavaScript, C#, and C++ instances (see Table II).

Our aim for the trained models is to classify the various instances in the dataset (i.e. sections obtained from readme files) into one of the following classes: Functionality, Usage, or Miscellaneous so that information required to construct the comparison charts can be obtained. The *Functionality* class represents instances about a repository containing statements such as an overview, high-level features or functionalities, a list of its advantages, how it compares to other solutions, and changes over different versions/releases. We consider that information in such sections highly relevant for constructing comparison charts. The *Usage* class represents instances corresponding to the how-to details, such as configuration, installation, and coding instructions related to a repository. Thus, these sections can provide additional information for constructing more detailed comparison charts. Finally, the *Miscellaneous* class represents instances corresponding to sections less relevant to comparison chart generation, such as user contributions and donations.

TABLE IV. Models Incorporated Into Lbl2TransformerVec During Training

Model	Ref./Model Card
bart-large-mnli	[23]
all-MiniLM-L6-v2	[24]
all-mpnet-base-v2	[25]
all-distilroberta-v1	[26]
all-MiniLM-L12-v2	[24]
unsup-simcse-bert-base-uncased	[27]
unsup-simcse-bert-large-uncased	[27]
unsup-simcse-roberta-base	[27]
unsup-simcse-roberta-large	[27]

To train the models, we pass the keywords list for each class to the Lbl2TransformerVec algorithm, shown in Table V. These keywords are chosen based on a combination of expertise and familiarity with GitHub readme files and by relying on a dictionary for word synonyms. For the Lbl2TransformerVec hyperparameters used to train the models, we set the *similarity_threshold* to 0.6 so that only instances with this threshold or higher with respect to the provided keywords are included to calculate the label embeddings. Furthermore, we set the *min_num_docs* parameter, which controls the minimum number of instances used to calculate the label embeddings, to 700. All other hyperparameters are set to their default values. Table VI provides the hyperparameter values used for training the various models.

After model training, to better understand their ability to classify the content of readme files, we run the models on the training dataset to classify all instances and then extract the most frequent words for each class. Fig. 3 plots the class-word distribution with a KxM shape, where K is 3, representing the number of classes, and M is the vocabulary size. As shown in this figure, the most frequent words for class 3 (i.e. sections classified by the model as Miscellaneous) include license, contributing, community, issues, and support, indicating the model's capability of labeling such sections. The most frequent words for sections labeled by the model as class 2 (i.e. Usage class) include *use*, *build*, *install*, and *run*. Finally, sections labeled with class 1 (i.e. Functionality) have as most frequent words the words *library*, *features*, *platform*, and *simple*, all of which are likely to appear in sections discussing the high-level functionalities and features of a repository. It can also be seen that several classes share some common frequent words such as *link* and *code*. This is because these words can appear in any section of these classes. For example, adding URLs pointing to external websites is a common practice for defining some terminologies or redirecting the user to extra resources to understand some functionality. Therefore, these URLs are part of Functionality. On the other hand, adding URLs in usage-related sections is also a common practice to refer users to more detailed installation documentation, configuration, and usage. Similarly, for the word *code*, readme files of many GitHub repositories can contain code snippets in the introduction, usage, and citation sections.

E. Rule-Based Classification Adjustments

During class prediction, the keyword-driven classifier may assign approximately similar scores to different classes for a section, which may result in incorrect classification for some of these sections. We incorporate rule-based adjustments to enhance classification results in such cases by considering the classes assigned to parent and sibling sections of a section i as follows.

Let i be a (sub)section in a readme file R, p^i be the predicted class for i, and p^i_2 be the second most likely class for i as scored by the keyword-driven classifier. For any

TABLE V. KEYWORDS USED FOR MODEL TRAINING

Class	Keywords	Purpose
Functionality	Introduction, Intro, Welcome, Overview, Index, Compatability, Comparison, What's, New, Motivation, Contents, Feature, Provide, Contain, Supports, Definition, Goal, Overview, Roadmap, Release, Version, Vision, About, What, About, Simple, Fast, Reliable, Flexible, Modern, Powerful, Cross-Platform, Alternative	(Sub)sections labeled by the model with this class are likely to contain high-level features and functionality of the repository
Usage	Getting, Started, Demo, Try, Updating, Quick, Start, Code, Snippet, Steps, Commands, Configuration, Setup, Requirements, Install, Uninstall, Installation, Tutorial, Instructions, Documentation, Dependencies, Prerequisites, Manual, Example, Examples, Resources, FAQ, Usage, How, Import, Flags, Parameters, Arguments, Download	(Sub)sections labeled by the model with this class are likely to contain detailed functionality and usage information related to the repository
Miscellaneous	Contribution, Contributing, Contribute, Contributed, Partners, Sponsors, Authors, Backers, BibTex, Community, Mission, Feedback, Copyright, Disclaimer, Trademark, Credits, Publications, Conduct, DOI, Thank, Thanks, Please, Announcements, Legal, Subscribe, Issues, Contact, Join, Inquiries, Donation, Donate, Citation, Cite, Paper, Licensed, License, Help, Support Discussion, Social, Twitter, Telegram, Facebook, Discord, Forum, Backers, Acknowledgment, Acknowledgments, Company, People, Who	(Sub)sections labeled by the model with this class are less likely to contain important features and functionality of the repository

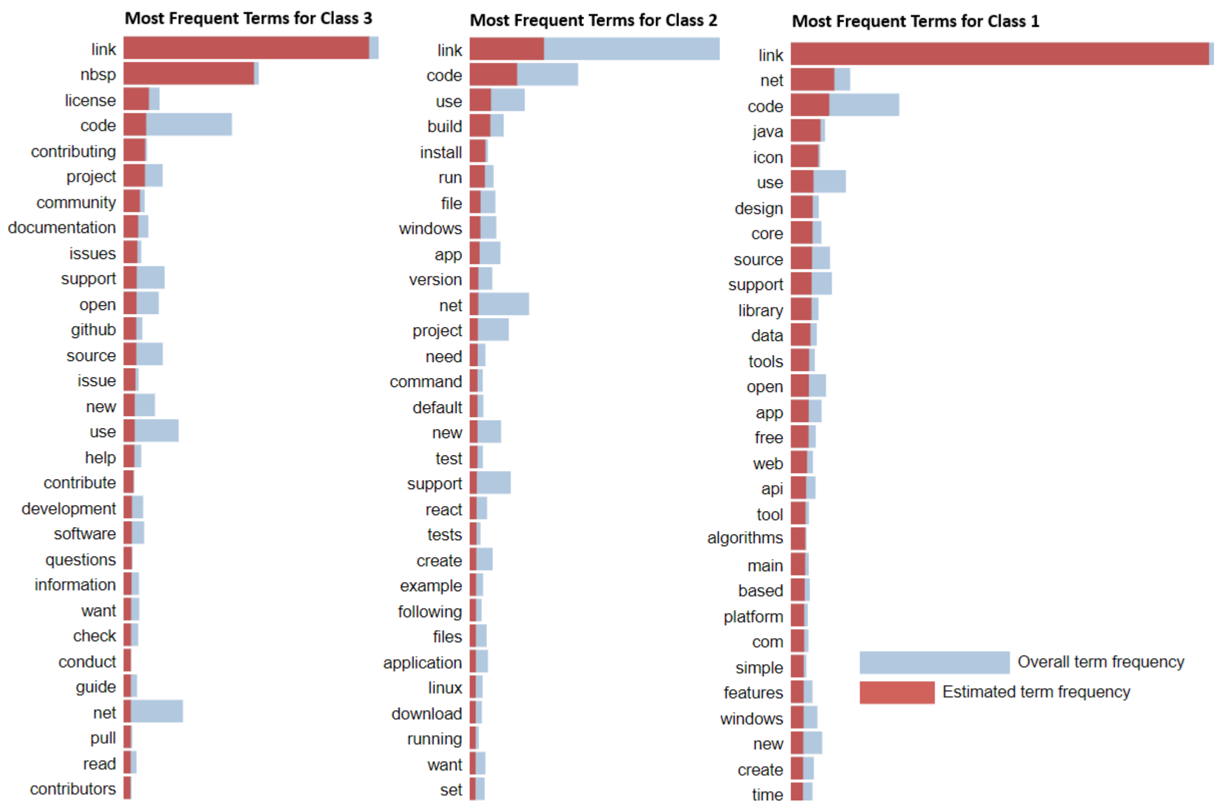


Fig. 3. Class-word distribution of training dataset for functionality (class 1), Usage (class 2), and miscellaneous (class 3).

TABLE VI. HYPERPARAMETERS PASSED TO LBL2TRANSFORMERVEC FOR MODEL TRAINING

Hyperparameter	Value
keywords_list	The keywords list shown in Table V
transformer_model	The models shown in Table IV
similarity_threshold	0.6
similarity_threshold_offset	default (0)
min_num_docs	700
max_num_docs	default (None)
clean_outliers	default (False)

(sub)section $i \in R$, if $|score(p^i) - score(p_2^i)| < \alpha$, then the following rules are applied, in the shown order, to adjust the predicted class for i:

1) Keywords ratio rule: The model computes the number of keywords (see Table V) that appeared in i for each class and then calculates the ratio of these numbers. If the ratio of class p_2^i is larger than a threshold β, then the predicted class of i is set to p_2^i.

2) Relevance to parent rule: The model considers the predicted class assigned to i's parent section. If (1) the parent section is classified as class p_2^i, (2) the parent section has a high keywords ratio for one class, (3) i has a low keywords ratio, and (4) α is negligible, then the predicted class of i is set to p_2^i.

3) Relevance to siblings rule: The model considers the predicted classes assigned to i's siblings (i.e. subsections at the same level as i) by calculating the class frequency of these siblings sections. If the most frequent class for these siblings is class p_2^i and this frequency exceeds a threshold β, then the predicted class of i is set to p_2^i.

To illustrate each rule and its purpose for adjusting classification, consider the examples shown in Table VII of erroneous classifications made by the keyword-driven model to some instances in the dataset:

- For instance 1, this instance discusses usage/documentation of the repository. However, the model classified this instance as *Miscellaneous*. Since the score difference between the actual and predicted classes is less than $\alpha = 0.04$ and this instance has more keywords related to the *Usage* class compared to other classes, then the model adjusts the predicted class for this instance from *Miscellaneous* to *Usage* according to the *keywords ratio* rule.
- For instance 2, this instance discusses usage details of a repository's functionality. However, the model classified this instance as *Features*. Since the score difference between the actual and predicted classes is less than $\alpha = 0.05$ and the model labeled the parent's section of this instance as *Usage*, then the model adjusts the predicted class for this instance from *Features* to *Usage* according to the *relevance to parent* rule.
- For instance 3, this instance provides users testimonials of the repository. However, the model classified this instance as *Functionality*. Since the score difference between the actual and predicted classes is less than $\alpha = 0.031$ and the most frequent class assigned by the model to sibling subsections is *Miscellaneous*, then the model adjusts the predicted class for this instance from *Functionality* to *Miscellaneous* according to the *relevance to siblings* rule.

F. Extraction of Repository Limitations

Repository owners may explicitly state in readme files the limitations of their repositories, such as unsupported features and functionalities or constraints related to operational environments. Such limitations must be identified so that they are not mistakenly classified as supported features by the proposed solution when generating comparison charts. Statements of such limitations in readme files may include specific keywords in sentences such as *limitation* or *unsupported*. Additionally, statements can include longer phrases to convey these limitations. For example, a repository of key-value storage may state that it does not support indexes or will receive limited maintenance. An example of such limitations is the readme file of Apache Airflow repository indicating that "MariaDB is not tested/recommended".

To extract these limitations and unsupported features of a repository, we incorporated a zero-shot classifier [28] based on the *bart-large-mnli* model, which is based on the *bart-large* model [23] and trained on the *MultiNLI* dataset consisting of 433k sentence pairs annotated with textual entailment information. For each section in the input readme files, we extract the sentences of the section and pass it to the zero-shot classifier to identify the score of being labeled as *"Unsupported Feature"* by the classifier. The default threshold is set to 0.9. Therefore, sentences that this zero-shot classifier scores with a value equal to or exceeding this threshold are extracted as candidate limitations for the repository.

G. Extraction of Repository Features

To identify a repository's supported features and capabilities, the multi-class classifier classifies extracted sections from the readme files corresponding to the user-selected *N* repositories. Sections labeled as *Functionaly* or *Usage* are further processed by removing all sentences corresponding to identified limitations by the zero-shot classifier. Then, we extract from these sections the keywords and key phrases capturing the repository's features and capabilities by incorporating KeyBERT [29]. The KeyBERT model is initialized with a Text-to-Text generation pipeline (also known as Sequence-to-Sequence modeling)[30]. We incorporate into this pipeline Llama-2-7b-chat-hf [31][32], which is a large langnuage model (LLM) consisting of 7 billion parameters fine-tuned for dialogue use cases. The pipeline tokenizes the provided text and relies on an encoder-decoder architecture to process the input text and generate a list of candidate features for each repository.

H. Calculating Similarity of Repositories' Features

Given each repository's extracted features, the online module calculates the semantic similarity between all possible feature pairs from the different repositories. The online module creates a sentence transformer [33] based on the sentence-t5-base [34] model to accomplish this task. For each feature pair of different repositories, the online module encodes the textual representation of each feature using the sentence transformer. As a result, the transformer encodes each feature into a 768-dimensional dense vector space. The cosine similarity [35] is then calculated from these vectors. Two features are labeled similar if the computed cosine similarity exceeds a threshold α.

IV. RESULTS

A. Evaluation Results of the Hybrid Classifier for Classifying the Content of GitHub Readme Files

We use the testing set corresponding to readme files obtained from GitHub repositories for the Python programming language to evaluate the hybrid classifier (see Table III). This set contains 984 instances and is not previously seen by the model since it was not used during training.

We first identify the *actual* class for each instance in the testing set through manual classification. Then, we run the hybrid model on this set to determine the *predicted* class for instances. Finally, we compute the F1 score to measure the predictive maintenance of the classifier. Although the proposed approach is unsupervised, manual classification is performed for evaluation purposes of the model.

To determine the actual class for each instance, we manually classify instances in the testing set by labeling each instance as either Functionality, Usage, or Miscellaneous. Our labeling process consists of reading the heading title of each (sub)section to determine its class. If we cannot determine the class from the subsection's heading title, then we read the first sentences in the subsection to determine its class. Finally, if

TABLE VII. Examples of Erroneous Classifications by the Keyword-driven Model and Explanation of Rule-based Adjustments

	Example Instance	Actual Class (Score by Model)	Predicted Class (Score by Model)	Explanation
1	Documentation. Read the Manual @link for more details.	Usage(0.5447)	Miscellaneous(0.5585)	Although the words *documentation*, *manual*, and *details* in this sentence correspond to keywords for class *Usage*, the model labeled this instance as *Miscellaneous*. Therefore, according to the *keywords ratio* rule, the predicted class is adjusted from Miscellaneous to *Usage*.
2	Train with DDL Statements. DDL statements contain information about the table names, columns, data types, and relationships in your database. @Code	Usage(0.6138)	Features(0.6187)	Although the model mistakenly labeled this instance as *Features* instead of *Usage*, this subsection is a child of a higher section that was labeled correctly by the model as *Usage*. Therefore, according to the *Relevance to Parent* rule, the predicted class is adjusted from Features to *Usage*.
3	Testimonials. Mike Bayer, author of SQLAlchemy link : I can't think of any single tool in my entire programming career that has given me a bigger productivity increase by its introduction. I can now do refactorings in about 1% of the keystrokes that it would have taken me previously when we had no way for code to format itself...	Miscellaneous(0.6173)	Functionality(0.6219)	Testimonial statements by users are labeled by the classifier as *Functionality* instead of *Miscellaneous* with a score difference of less than 0.01. However, the model correctly labels sibling subsections as *Miscellaneous*. Therefore, according to the *Relevance to Siblings* rule, the predicted class is adjusted from Functionality to *Miscellaneous*.

the class still cannot be determined, we read the content of the subsection to determine its class.

Given the actual and predicted classes for each instance in the test dataset, we compute the F1 score (also known as the balanced F-score) [36] according to the following formula:

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{1}$$

Where TP represents the number of true positives, FN represents the number of false negatives, and FP represents the number of false positives.

Evaluation results for the various Lbl2TransformVec models are shown in Table VIII. Classification based on keyword-driven approaches produces F1 scores that range from 0.6930 to 0.7601. Furthermore, several models have equal scores, possibly due to these models sharing the same base model. The *improvement* column shows the percentage change between the F1 scores obtained from the evaluation of keyword-driven models and the F1 scores obtained from the evaluation of combining keyword-driven models with rule-based adjustments. As seen in this table, incorporating rule-based adjustments shows observable improvement in scores by an average of 13.26% increase. Table IX shows an example of the contribution of each rule in adjusting the classification results, where the *keyword ratio rule* contributed the most by correcting 75 instances while failing to adjust 27 instances. On the other hand, the *relevance to the parent* rule has contributed the least in adjusting classification results. As seen in Table VIII rule-based adjustments contributed the least when applied to all-MiniLM-L6-v2. Inspection of correction results for this model reveals significant incorrect adjustments, particularly to the *relevance to siblings* rule with 40 incorrect adjustments.

We analyzed the class-word distribution of the classified instances from the testing set. The most frequent words for subsections identified as Functionality include words such as *models*, *data*, *index*, and *new*. The words model and data appear as frequently since many Python repositories discuss machine-learning-related libraries and algorithms. On the other hand, instances classified as Usage contain as frequent words the words *Python*, *install*, *use*, *run*, *command*, and *pip*. Finally, the words *license*, *contributing*, *issues*, *community*, and *help* appeared among the most frequent keywords in instances classified as Miscellaneous.

TABLE VIII. F1 Scores for Content Classification using the Lbl2TransformerVec Algorithm with Different Underlying Models, with and without Rule-based Classification Adjustments

Model	F1 Score		Improv.
	Key-Driven	w/ Rule-Based	
bart-large-mnli	0.7601	**0.8607**	+13.23%
all-MiniLM-L6-v2	0.6961	**0.7571**	+8.76%
all-mpnet-base-v2	0.6930	**0.7957**	+14.82%
all-distilroberta-v1	0.6930	**0.7957**	+14.82%
all-MiniLM-L12-v2	0.6930	**0.7957**	+14.82%
unsup-simcse-bert-base-uncas	0.7601	**0.8607**	+13.23%
unsup-simcse-bert-large-uncas	0.7601	**0.8607**	+13.23%
unsup-simcse-roberta-base	0.7601	**0.8607**	+13.23%
unsup-simcse-roberta-large	0.7601	**0.8607**	+13.23%

TABLE IX. Examples of Correction Results by Rule-based Classification Adjustments

Rule	Correct Adjustments	Incorrect Adjustments
Keywords Ratio	75 instances	27 instances
Relevance to Parent	10 instances	5 instances
Relevance to Siblings	42 instances	18 instances

B. Evaluation Results of Zero-Shot Classifier for Classifying the Limitations of Repositories

To evaluate the zero-shot classifier presented in Section III-F, we constructed a subset dataset derived from the original dataset (see Table III) by searching for instances that explicitly state limitations of repositories as well as instances of non-limitation sentences. The resulting dataset consists of 55 instances, where 26 represent limitation sentences, and 29 represent non-limitation sentences. We run the zero-shot classifier to classify each instance in the subset dataset and then calculate the F1 score. Evaluation results show that the predictive performance of the zero-shot classifier in classifying sentences as being *Unsupported Features* has an F1 score of 0.72.

Table X shows examples of limitation sentences found in several instances in the dataset, with some instances of non-limitation sentences. As can be seen in this table, the classifier gave high scores to sentences including specific keywords and key phrases (such as *"not tested/recommended", "won't be able to save files", "does not work on" and "not compatible with"*. On the other hand, instances 5 and 6, which do not convey any limitations, scored very low by the zero-shot classifier, as expected. Instances 7-9 show examples of erroneous classifications by the zero-shot classifier in nuanced cases where limitations can be implied.

C. Motivation Example Continued: Generated Comparison Charts

Continuing with the motivation example discussed in Section I, Fig. 4 shows the result of generating the comparison chart, represented as an HTML file, by the online module for this example. In this example, the user selects $N = 3$ repositories from GitHub's search results. These repositories are Mailspring [37], Mailpile [38], and emailengine [39], all of which are top-starred repositories for the search term "email client". The comparison chart displays the features of each repository. If two features from different repositories are similar, they are assigned an equal number (shown in blue in Fig. 4). The identified limitations for each repository (if any) are shown next.

In this chart, the online module has identified a single limitation for the second repository, which corresponds to the Mailpile repository, as obsolete. Inspection of the repository's readme file reveals that the online module has identified this limitation through the zero-shot classifier since it appeared in the introduction section of the file, where the Lbl2TransformerVec algorithm previously labeled this section with the feature class.

The lists of features and extended features in this comparison chart are extracted from *Functionality* and *Usage* sections, respectively. The total number of words in this chart's features and extended features lists is 397. Compared to the total number of words in the readme files for the three repositories, which is 1742 words, the comparison chart achieves a reduction of 125% of textual information that needs to be read by the user.

The commonality between repositories based on semantic similarity is shown in the features list. For each pair of similar features, a unique number (shown in blue in Fig. 4) is assigned. For example, the Mailspring and Mailpile repositories include "fast" as a feature. As a result, both features are identified by the online module as common. Similarly, both repositories indicate that they are free and are grouped as similar features. However, our results include false positives. For example, the "Read Receipt" and "Documentation and Details" are calculated as similar.

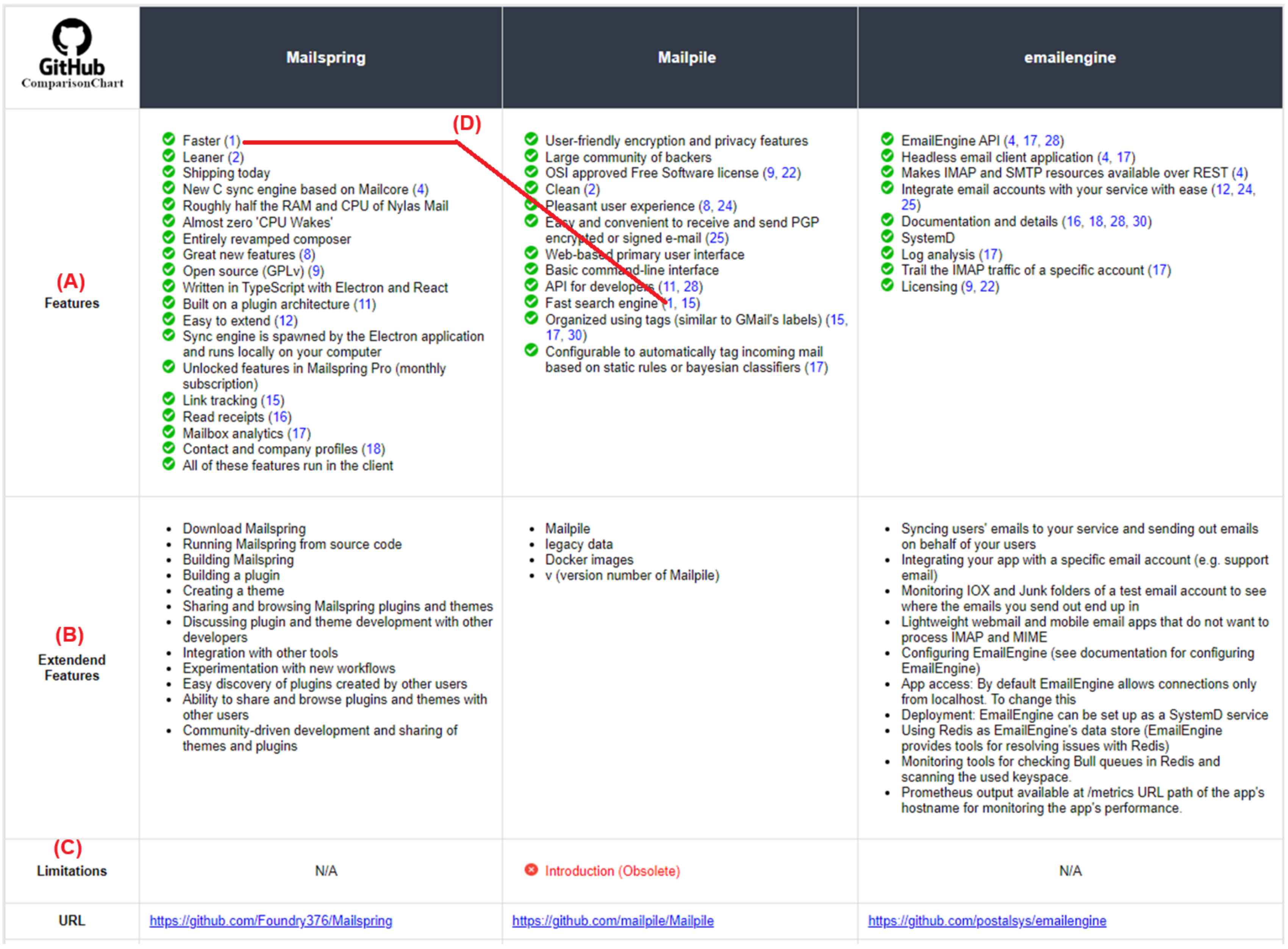
V. Discussion and Threats to Validity

Analyzing the readme files of similar GitHub repositories to understand their unique features, limitations, and similarities with one another is a challenging task since these files do not adhere to a standard and may vary in their level of detail. Furthermore, these files are written in different styles. Our results show that combining several Large Language Models (LLMs) can address these challenges and generate useful comparison charts for these repositories. First, the keyword-driven classifier based on Lbl2TransformerVec can identify relevant sections (containing important information about the repository) and irrelevant sections (that are unlikely to contain significant information about the repository's provided features, such as Contribution Acknowledgment, Licence, and Donation). After identifying relevant sections, their sentences are extracted and passed to a zero-shot LLM based on *bart-large-mnli* to determine whether the sentence intends to convey a limitation of the repository. These limitations are extracted so that they are not mistakenly considered as features by the proposed approach. A keyword extractor LLM based on KeyBERT is incorporated to extract the most relevant keywords representing the features of each repository. Finally, a sentence transformer is incorporated to find the semantic similarity between features of different repositories.

Although our results show that models generated by Lbl2TransofmerVec, which is a similarity-based approach leveraging embeddings generated by deep learning models [5], can classify the content of readme files, rule-based adjustments can improve the algorithm's results. This is because Lbl2TransofmerVec trains models to classify (sub)sections in isolation irrespective of their relations to other sections. Therefore, the lack of such view of relations during model training can cause incorrect classifications by these models. The rule-based adjustments complement keyword classification with such a view where these relations are considered. These rule-based adjustments are possible since readme files are often well-structured as reported previously by Treude et al. [16].

The collected dataset covers different application domains since the collection process is blind to such domains. To confirm this, we inspected the collected Python repositories used for testing to determine their domains. Inspection reveals that the test dataset covers various domains such as video production, deepfake, cryptocurrencies, cloud development, SQL-related libraries, and machine learning. Therefore, the proposed approach generalizes across different repository types or domains as it relies on textual readme files unrelated to the source code or bytecode of repositories [13].

Our results show that larger sequence-to-sequence and bidirectional transformer encoder models such as bart-large-mnli and unsup-simcse variants achieve better F1 scores than smaller models such as all-MiniLM-L6-v2. This is because larger models with more parameters and larger embeddings can capture complex patterns and relationships within the data compared to smaller models with fewer parameters and smaller embeddings. On the other hand, we observe that smaller models, such as all-MiniLM-L6-v2, are more efficient than larger models in terms of training and inference times and require lower resource usage, making them more appropriate when dealing with resource-constrained environments.



GitHub ComparisonChart	Mailspring	Mailpile	emailengine
(A) Features	✓ Faster (1) ✓ Leaner (2) ✓ Shipping today ✓ New C sync engine based on Mailcore (4) ✓ Roughly half the RAM and CPU of Nylas Mail ✓ Almost zero 'CPU Wakes' ✓ Entirely revamped composer ✓ Great new features (8) ✓ Open source (GPLv) (9) ✓ Written in TypeScript with Electron and React ✓ Built on a plugin architecture (11) ✓ Easy to extend (12) ✓ Sync engine is spawned by the Electron application and runs locally on your computer ✓ Unlocked features in Mailspring Pro (monthly subscription) ✓ Link tracking (15) ✓ Read receipts (16) ✓ Mailbox analytics (17) ✓ Contact and company profiles (18) ✓ All of these features run in the client	(D) ✓ User-friendly encryption and privacy features ✓ Large community of backers ✓ OSI approved Free Software license (9, 22) ✓ Clean (2) ✓ Pleasant user experience (8, 24) ✓ Easy and convenient to receive and send PGP encrypted or signed e-mail (25) ✓ Web-based primary user interface ✓ Basic command-line interface ✓ API for developers (11, 28) ✓ Fast search engine (1, 15) ✓ Organized using tags (similar to GMail's labels) (15, 17, 30) ✓ Configurable to automatically tag incoming mail based on static rules or bayesian classifiers (17)	✓ EmailEngine API (4, 17, 28) ✓ Headless email client application (4, 17) ✓ Makes IMAP and SMTP resources available over REST (4) ✓ Integrate email accounts with your service with ease (12, 24, 25) ✓ Documentation and details (16, 18, 28, 30) ✓ SystemD ✓ Log analysis (17) ✓ Trail the IMAP traffic of a specific account (17) ✓ Licensing (9, 22)
(B) Extendend Features	• Download Mailspring • Running Mailspring from source code • Building Mailspring • Building a plugin • Creating a theme • Sharing and browsing Mailspring plugins and themes • Discussing plugin and theme development with other developers • Integration with other tools • Experimentation with new workflows • Easy discovery of plugins created by other users • Ability to share and browse plugins and themes with other users • Community-driven development and sharing of themes and plugins	• Mailpile • legacy data • Docker images • v (version number of Mailpile)	• Syncing users' emails to your service and sending out emails on behalf of your users • Integrating your app with a specific email account (e.g. support email) • Monitoring IOX and Junk folders of a test email account to see where the emails you send out end up in • Lightweight webmail and mobile email apps that do not want to process IMAP and MIME • Configuring EmailEngine (see documentation for configuring EmailEngine) • App access: By default EmailEngine allows connections only from localhost. To change this • Deployment: EmailEngine can be set up as a SystemD service • Using Redis as EmailEngine's data store (EmailEngine provides tools for resolving issues with Redis) • Monitoring tools for checking Bull queues in Redis and scanning the used keyspace. • Prometheus output available at /metrics URL path of the app's hostname for monitoring the app's performance.
(C) Limitations	N/A	✗ Introduction (Obsolete)	N/A
URL	https://github.com/Foundry376/Mailspring	https://github.com/mailpile/Mailpile	https://github.com/postalsys/emailengine

Fig. 4. An example of a generated comparison chart for three GitHub repositories. In this chart, the features (A) and extended features (B) of repositories are identified by KeyBert from sections that are classified as *Functionality* and *Usage*, respectively. Limitations (C) are sentences identified by the zero-shot classifier as *Unsupported Features*. Features from different repositories with matching numbers (D) are considered as potentially common.

TABLE X. Examples of Zero-shot Classification of Limitations where Score Threshold is Set to 0.9

No.	Instance Example	Repository	Is Limitation?		Score
			Predicted	Expected	
1	MariaDB is not tested/recommended	Apache Airflow	Yes	Yes	0.997
2	You won't be able to save files to system folders due to UWP restriction windows, system32	Notepads	Yes	Yes	0.995
3	It does not work on non-Android devices incl. LG or Samsung TVs	SmartTube	Yes	Yes	0.94
4	Swiper is not compatible with all platforms	Swiper	Yes	Yes	0.986
5	If your platform is unsupported or not listed above, there is still a chance you can run the release or manually build it by following the instructions	osu	No	No	0.33
6	it is a modern touch slider which is focused only on modern apps/platforms to bring the best experience and simplicity	Swiper	No	No	0.001
7	Importantly, we have not yet fine-tuned the Alpaca model to be safe and harmless	stanford alpaca	No	Yes	0.28
8	This repository is receiving very limited maintenance	leveldb	No	Yes	0.11
9	convert.py has been deprecated and moved to examples/convert-legacy-llama.py, please use convert-hf-to-gguf.py @link	llama.cpp	Yes	No	0.99

Compared to prior works incorporating supervised learning for classifying the content of GitHub readme files, Perna et al. reported an F1 score of 0.72 [2], while our hybrid approach achieved an F1 score of up to 0.86 (with rule-based adjustment). However, it should be noted that their supervised approach is multi-label and considers more classes. In contrast, our approach merges some of the classes they reported in their work as we focus on extracting the functionality and features of repositories from readme files. For example, the What, Why, and When classes in their work correspond to the Functionality class in our work.

Although our results show adequate extraction of comparison charts, we identify several challenges and possible improvements for future works as follows:

- *Immature or incomplete readme files*: Throughout our data collection process, we discovered that the readme files of many GitHub repositories, including repositories with high star ratings, may lack detailed information on their functionalities and features. Instead, they add internal or external URLs (such as the product's official website) for further details. Therefore, the generated comparison charts are incomplete and do not represent the repositories' full features and limitations. Thus, the proposed approach can be extended to automatically cover such internal and external resources. In addition to analyzing the readme files of a repository, it is also possible to extend the proposed approach by gathering additional sources of information using GitHub's API, including GitHub issues, pull requests, and discussions. A pull request represents a proposal for merging a set of changes before these changes are integrated into the main codebase [40]. Therefore, analyzing pull requests and their current statuses makes it possible to expand the comparison charts with information unavailable in readme files. For example, merged pull requests can reveal new features/functionalities of a repository. Similarly, an open pull request can reveal potential limitations yet to be addressed. GitHub issues and discussions can be analyzed to discover a repository's features and limitations. However, since any user can create issues and participate in public repositories, the challenge is to validate these issues and discussions before relevant information is extracted and used in comparison charts, as some issues can result from user misunderstanding or misuse of the repository.
- *Sentence-Level content classification*: Our work assumes that each (sub)section in readme files is mapped to a specific class (e.g. functionality, usage, or Miscellaneous). Although many repository owners ensure that each (sub)section has a clear and single purpose to achieve, our observation indicates sections can include sentences of various classes. For example, some readme files may contain Usage instructions in the introduction section. This may result in missed features and functionalities if conveyed in sections classified as Miscellaneous. Therefore, one may consider enhancing the classification task at the sentence level for such sections.
- *Interpretability and explainability*: The generated comparison charts can be inaccurate and biased. Therefore, it is essential to incorporate appropriate mechanisms so that these charts are explainable [41] [42] to the end users, conveying the underlying model's accuracy and achieving transparency. For example, interactive elements can be added to these charts to explain how the proposed approach obtains and calculates the various parts (i.e. repository features, extended features, limitations, and similarities). Furthermore, incorporating tractability mechanisms between these parts and the source from which they are obtained (i.e. line numbers within readme files) enables users to validate these parts.
- *Repository Limitations Extraction*: Our evaluation results of the zero-shot classifier to identify the limitations of repositories demonstrate its capability for this task. However, this approach needs to be investigated further. First, the small dataset used to validate the zero-shot classifier is a threat to validity, as a larger dataset is required for evaluation. The challenge is obtaining a large dataset representing limitations found in real GitHub repositories, as most repository owners focus on stating what their repository provides rather than stating the limitations. Second, our results show that sentences with implied limitations (e.g. instance 7 in Table X) can result in erroneous classifications by the zero-shot classifier. One potential solution is to train the classifier on a dataset of real limitations from GitHub repositories. Finally, the zero-shot classifier lacks a contextual view during classification;

for example, the classifier cannot distinguish whether the limitation is related to the user-selected repository or other related/external repositories referenced in a readme file. This challenge applies to identifying a repository's features as well. A possible improvement is to add steps for analyzing the sentence(s) structure and linguistic features to determine whether a limitation (or a feature) is related to the user-selected repository and not other related repositories referenced in the readme files.

- *Feature commonalities between repositories*: Our results show that identifying common features between different repositories is challenging. One possible reason is the usage of technical terms that are not incorporated during the training of LLMs. One potential future direction is to investigate the enhancement of similarity approaches in this respect.
- *Quantitative Metrics Generation*: Our approach relies on extracting qualitative data from readme text files. However, it might be possible to extract more qualitative and quantitative data. For example, the frequency with which each repository is updated, the average response time of a repository's owner to issues, and the overall maturity of each repository compared to others can be considered. These metrics can enrich the generated comparison charts and their overall added value to end-users.

VI. Conclusion

While prior works investigated the analysis of individual readme files, with most works following a supervised approach, our work focuses on an unsupervised approach for the classification task, which aims to generate comparison charts of similar GitHub repositories from their readme files. Our evaluation results show that textual information in comparison charts can be 125% less compared to the amount of text in readme files, thus minimizing the time and effort required for users to read these files to understand and compare their features and capabilities. Our approach utilizes a hybrid model based on the Lbl2TransformerVec algorithm and augmented with rule-based classification adjustments. The model is trained based on readme files automatically obtained from GitHub for Java, JavaScript, C++, and C# repositories. The model is then evaluated using a different set of readme files from Python repositories. Our results show that rule-based classification adjustment can improve the model's predictive performance by up to 13%. We then incorporated this model in an online module to generate comparison charts of GitHub repositories based on user search terms. Our future work includes investigating several enhancements to the proposed approach to address its limitations, as identified in the previous section, including (1) investigating how comparison charts can be enriched with information from other sources (such as GitHub pull requests, discussions, and issues), (2) extending the proposed approach so that the generated comparison charts are explainable to the end users, and (3) adding quantitative metrics to these charts. Furthermore, we plan to expand the evaluation of this approach using a larger dataset of GitHub repository features and limitations.

References

[1] GitHub, Inc. (2024) Build software better, together. [Online]. Available: https://github.com/about

[2] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the Content of GitHub README Files," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1296–1327, Jun. 2019. [Online]. Available: https://doi.org/10.1007/s10664-018-9660-3

[3] J. Wu, Y. Sun, and J. Zhang, "An Open-source Repository Retrieval Service Using Functional Semantics for Software Developers," in *2022 International Conference on Service Science (ICSS)*, May 2022, pp. 12–20. [Online]. Available: https://ieeexplore.ieee.org/document/9860185

[4] Y. Zhang, F. F. Xu, S. Li, Y. Meng, X. Wang, Q. Li, and J. Han, "Higitclass: Keyword-driven hierarchical classification of github repositories," in *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, J. Wang, K. Shim, and X. Wu, Eds. IEEE, 2019, pp. 876–885. [Online]. Available: https://doi.org/10.1109/ICDM.2019.00098

[5] T. Schopf, D. Braun, and F. Matthes, "Evaluating unsupervised text classification: Zero-shot and similarity-based approaches," in *Proceedings of the 2022 6th International Conference on Natural Language Processing and Information Retrieval*, ser. NLPIR '22. New York, NY, USA: Association for Computing Machinery, 2023, p. 6–15. [Online]. Available: https://doi.org/10.1145/3582768.3582795

[6] T. Schopf, D. Braun, and F. Matthes, "Semantic label representations with lbl2vec: A similarity-based approach for unsupervised text classification," in *Web Information Systems and Technologies*, ser. Lecture Notes in Business Information Processing, M. Marchiori, F. Domínguez Mayo, and J. Filipe, Eds. Germany: Springer, Jan. 2023, pp. 59–73.

[7] T. Schopf, D. Braun, and F. Matthes, "Lbl2vec: An embedding-based approach for unsupervised document retrieval on predefined topics," in *Proceedings of the 17th International Conference on Web Information Systems and Technologies - WEBIST,*, INSTICC. SciTePress, 2021, pp. 124–132.

[8] Y. Zhou, J. Wu, and Y. Sun, "GHTRec: A Personalized Service to Recommend GitHub Trending Repositories for Developers," in *2021 IEEE International Conference on Web Services (ICWS)*, Sep. 2021, pp. 314–323. [Online]. Available: https://ieeexplore.ieee.org/document/9590294

[9] C. Di Sipio, R. Rubei, D. Di Ruscio, and P. T. Nguyen, "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*, ser. EASE '20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 71–80. [Online]. Available: https://doi.org/10.1145/3383219.3383227

[10] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo, "Cataloging github repositories," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 314–319. [Online]. Available: https://doi.org/10.1145/3084226.3084287

[11] F. Zanartu, C. Treude, B. Cartaxo, H. S. Borges, P. Moura, M. Wagner, and G. Pinto, "Automatically categorising github repositories by application domain," 2022. [Online]. Available: https://arxiv.org/abs/2208.00269

[12] S. Vargas-Baldrich, M. Linares-Vásquez, and D. Poshyvanyk, "Automated tagging of software projects using bytecode and dependencies (n)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 289–294.

[13] M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," *Empirical Software Engineering*, vol. 26, no. 5, p. 93, Jul. 2021. [Online]. Available: https://doi.org/10.1007/s10664-021-09976-2

[14] Y. Liu, E. Noei, and K. Lyons, "How ReadMe files are structured in open source Java projects," *Information and Software Technology*, vol. 148, p. 106924, Aug. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922000775

[15] A. S. M. Venigalla and S. Chimalakonda, "An empirical study on correlation between readme content and project popularity," 2022. [Online]. Available: https://arxiv.org/abs/2206.10772

[16] C. Treude, J. Middleton, and T. Atapattu, "Beyond accuracy: assessing software documentation quality," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 1509–1512. [Online]. Available: https://doi.org/10.1145/3368089.3417045

[17] O. Elazhary, M.-A. Storey, N. Ernst, and A. Zaidman, "Do as I Do, Not as I Say: Do Contribution Guidelines Match the GitHub Contribution Process?" *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 286–290, Sep. 2019, conference Name: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) ISBN: 9781728130941 Place: Cleveland, OH, USA Publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/document/8919187/

[18] A. S. M. Venigalla and S. Chimalakonda, "What's in a github repository? – a software documentation perspective," 2021. [Online]. Available: https://arxiv.org/abs/2102.12727

[19] J. Hellman, E. Jang, C. Treude, C. Huang, and J. L. C. Guo, "Generating github repository descriptions: A comparison of manual and automated approaches," 2021. [Online]. Available: https://arxiv.org/abs/2110.13283

[20] Y. Zhou, J. Wu, and Y. Sun, "Ghtrec: A personalized service to recommend github trending repositories for developers," in *2021 IEEE International Conference on Web Services (ICWS)*, 2021, pp. 314–323.

[21] GitHub, Inc. REST API endpoints for search. [Online]. Available: https://docs.github.com/en/rest/search/search

[22] GitHub, Inc. Writing on GitHub. [Online]. Available: https://docs.github.com/en/get-started/writing-on-github

[23] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds. Online: Association for Computational Linguistics, Jul. 2020, pp. 7871–7880. [Online]. Available: https://aclanthology.org/2020.acl-main.703

[24] W. Wang, H. Bao, S. Huang, L. Dong, and F. Wei, "MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 2140–2151. [Online]. Available: https://aclanthology.org/2021.findings-acl.188

[25] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "Mpnet: masked and permuted pre-training for language understanding," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[26] sentence-transformers/all-distilroberta-v1 · Hugging Face. [Online]. Available: https://huggingface.co/sentence-transformers/all-distilroberta-v1

[27] T. Gao, X. Yao, and D. Chen, "SimCSE: Simple contrastive learning of sentence embeddings," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[28] W. Yin, J. Hay, and D. Roth, "Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3914–3923. [Online]. Available: https://aclanthology.org/D19-1404

[29] M. Grootendorst, "Keybert: Minimal keyword extraction with bert." 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4461265

[30] Hugging face pipelines. [Online]. Available: https://huggingface.co/docs/transformers/en/main_classes/pipelines

[31] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[32] (2024, Aug.) meta-llama/Llama-2-7b-chat-hf · Hugging Face. [Online]. Available: https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

[33] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[34] J. Ni, G. Hernandez Abrego, N. Constant, J. Ma, K. Hall, D. Cer, and Y. Yang, "Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models," in *Findings of the Association for Computational Linguistics: ACL 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1864–1874. [Online]. Available: https://aclanthology.org/2022.findings-acl.146

[35] E. Schubert, "A triangle inequality for cosine similarity," in *Similarity Search and Applications*, N. Reyes, R. Connor, N. Kriege, D. Kazempour, I. Bartolini, E. Schubert, and J.-J. Chen, Eds. Cham: Springer International Publishing, 2021, pp. 32–44.

[36] Z. C. Lipton, C. Elkan, and B. Naryanaswamy, "Optimal thresholding of classifiers to maximize f1 measure," in *Machine Learning and Knowledge Discovery in Databases*, T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 225–239.

[37] (2024, Sep.) Foundry376/Mailspring. Original-date: 2016-10-13T06:45:50Z. [Online]. Available: https://github.com/Foundry376/Mailspring

[38] (2024, Sep.) mailpile/Mailpile. Original-date: 2011-10-30T23:45:22Z. [Online]. Available: https://github.com/mailpile/Mailpile

[39] (2024, Sep.) postalsys/emailengine. Original-date: 2020-02-28T17:17:44Z. [Online]. Available: https://github.com/postalsys/emailengine

[40] GitHub, Inc. About pull requests. [Online]. Available: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

[41] N. A. Sharma, R. R. Chand, Z. Buksh, A. B. M. S. Ali, A. Hanif, and A. Beheshti, "Explainable AI Frameworks: Navigating the Present Challenges and Unveiling Innovative Applications," *Algorithms*, vol. 17, no. 6, 2024. [Online]. Available: https://www.mdpi.com/1999-4893/17/6/227

[42] S. A. and S. R., "A systematic review of explainable artificial intelligence models and applications: Recent developments and future trends," *Decision Analytics Journal*, vol. 7, p. 100230, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S277266222300070X