International Journal of Information Security (2025) 24:158
https://doi.org/10.1007/s10207-025-01069-1

REGULAR CONTRIBUTION

Identifying Open-Source Threat Detection Resources on GitHub: A Scalable Machine Learning Approach

Manuel Kern[1] · Max Landauer[1] · Florian Skopik[1] · Edgar Weippl[2]

Abstract
Many businesses rely on open-source software modules to build their technology stacks. However, those who lack domain expertise may struggle to find the right software due to unfamiliar terminology and specific names. As a consequence, search engines and other platforms often cannot be utilized effectively to discover appropriate solutions. There is thus a need for a more applicable approach to assist non-domain experts in navigating the vastness of available repositories, enabling them to efficiently discover and select the right solution for their business needs. To overcome these gaps, we introduce an approach that supports finding unpopular yet important open-source software repositories on GitHub using advanced machine learning techniques. For this purpose, we propose novel strategies for information gathering and data pre-processing that resolve scalability issues of existing solutions and enable clustering of repositories even when topics, descriptions, or repository names are unclear or absent. For our evaluation, we gathered a dataset of 221,971 repositories using GitHub search and keywords related to incident detection. We show that our approach is able to separate threat detection repositories from others with an F1-score of 0.93.

Keywords Threat detection · Usable security · Machine learning · Repository discovery · Topic clustering

1 Introduction

In the last decade, open source software has become an increasingly important factor in the IT world. Nowadays, many software applications and frameworks that are or at least partially base on open source software are widely used in corporate, public, and academic domains, such as the Android operating system for mobile phones, the Openstack cloud computing platform, the Hadoop framework for big data processing, etc. [1]. Multiple large websites have been established to enable collaboration and development as well as distribution of open source software. Currently, the biggest player on the market is arguably GitHub, but many alternative providers such as Sourceforge, CodeCommit, BitBucket, GitLab, and Google Cloud Repositories recently gained in popularity.

Possibly as a direct consequence of this issue, the recent trend of so-called "awesome" lists, i.e., community-curated lists of links to resources such as GitHub repositories relevant for a specific application domain, has evolved. One of the largest awesome lists hosted on GitHub is from user sindreshorhus[1] and lists more than 500 contributors. This list holds links to other GitHub lists in domains such as programming languages, computer science, big data, entertainment, security, etc., in a structured way.

While awesome lists are a convenient method to obtain an overview of available projects in common domains, the problem to find new or less popular repositories remains. In 2017, GitHub integrated so-called topics that allow developers to categorize their repositories and thereby make it easier for users to find different repositories belonging to

✉ Manuel Kern
manuel.kern@ait.ac.at

Max Landauer
Max.Landauer@ait.ac.at

Florian Skopik
Florian.Skopik@ait.ac.at

Edgar Weippl
Edgar.Weippl@univie.ac.at

[1] AIT Austrian Institute of Technology, Giefinggasse 4, 1210 Vienna, Austria

[2] University of Vienna, Universitaetsring 1, 1010 Vienna, Austria

[1] https://github.com/sindresorhus/awesome

Published online: 17 June 2025

the same domain. While GitHub allows to sort repositories retrieved for certain topics by criteria such as "most stars", "fewest stars", "most forks", "fewest forks", "recently updated", and "least recently updated", it still remains difficult to find certain repositories as many topics are used scarcely, while others are too generic [2]. For example, the topic called encryption[2] contains over 7,000 repositories (as of Sept. 2023), making it hardly possible to identify repositories that address a specific problem within the encryption domain, especially if they are not very popular (i.e., have few stars or forks) and not very active anymore (i.e., have not recently been updated). However, the most influential factor with respect to incomplete results obtained from searches by topic is that GitHub does not require to assign topics to repositories, in which case the project cannot be found at all. In course of our experiments we observed that even among widely popular software projects topics are often missing, e.g., the widely deployed IDS Snort[3] lacks topics. The problematic situation with topics is further aggravated by the lack of a unified taxonomy. Specifically, topics are often synonyms or subsets of each other, which drastically limits their use for classification and retrieval.

An alternative method to find relevant repositories is the GitHub provided search. By default, the query keywords used are only matched on the about, title, and topic section of a repository. Similar to topics, repositories that lack the fully optional about section cannot be found through this way. In course of our research we identified that 68% of the repositories that were returned during our search do not provide topics and 12% have empty about sections. In contrast, only 1% of the repositories do not include a readme section, which is thus a promising source of information for retrieval of relevant repositories. In contrast to other approaches, we also utilize the directory tree and the names of the repositories to extract keywords, that we later use to find similar repositories even if they do not have any about section or topics defined.

Our experiments presented in this paper focus on GitHub repositories for cyber security threat detection. Several popular security applications such as intrusion detection systems are widely used in the corporate sector and available as open source software, while at the same time many forks of these repositories as well as independently developed smaller projects and niche repositories exist. In addition, security repositories are also commonly used to store artifacts other than software, such as tool configurations, detection rules, best practices, and so on. This allows us to collect a diverse dataset of 221,971 repositories comprising many similar and dissimilar projects that is suitable for evaluating our approach. To the best of our knowledge, this is the largest dataset that has been used to evaluate recommender systems for GitHub repositories. Moreover, this paper extends the state-of-the-art by proposing novel strategies to pre-process artifacts such as readme and about, name, tree and programm languages sections in addition to topics. As we show in the paper, this does not only enable effective retrieval of similar repositories with machine learning methods such as HDBSCAN and K-Means, but also addresses the problem of scalability when applying existing approaches such as BERTopic [3]. We provide the generated dataset and the code to run our experiments on GitHub[4]. We summarize the contributions of this paper as follows:

1. An investigation of available artifacts and auxiliary information sources in open source repositories,
2. a method to pre-process relevant artifacts and apply machine learning techniques to retrieve similar repositories independent from their popularity, and
3. an assessment of the quality of search results for open source software in the security domain.

The remainder of this paper is structured as follows. Sect. 2 discusses related works in the research area of repository filtering and discovery. Sect. 3 provides a conceptual overview of our proposed approach. We outline our methodology for collecting a dataset of repositories for experimentation and evaluation in Sect. 4. Sect. 5 contains an in-depth explanation of our proposed procedure and Sect. 6 presents the results we obtain from applying our approach on the data. We discuss the results and provide ideas for future work in Sect. 7. Finally, Sect. 8 concludes this paper.

2 Related work

A considerable amount of research is dedicated to exploring GitHub with data mining techniques. Di Rocco et al. [4] introduced TopFilter, a tool that aids developers in choosing appropriate topics for their repositories on GitHub. The TopFilter algorithm employs a graph-based representation of repositories to identify similar repositories, which are then used to suggest relevant topics for a specific repository. TopFilter builds upon Multinomial Naive Bayesian Networks (MNBN) [5], which suggests topics based on readme files but is limited to pre-defined topics and assumes that datasets are balanced. Both approaches were evaluated using the same datasets, with the largest dataset holding 13,400 repositories[5].

Repo-Topic[6] is a feature provided by GitHub that allows to extract topics using lightweight NLP techniques such as

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

[3] https://github.com/snort3/snort3

[4] https://github.com/ghml23/ghml23

[5] https://github.com/MDEGroup/MNB_TopicRecommendation/

[6] https://github.blog/2017-07-31-topics/

tf-idf. On the other hand, GHTRec [6] suggests personalized trending repositories using deep learning to predict topics by leveraging historical commits and trending repositories. In their latest work, Rocco et al. [7] have presented a new method called HybridRec. This approach utilizes a stochastic network and a collaborative filtering technique to suggest topics, while also considering readme files, Wiki contents, and commit messages. Repopal [8] shows similarities of repositories based on readme files and user activities. Widyasari et al. [2] utilized extreme multi-label learning on a dataset consisting of 21,000 repositories. They discovered that previous studies tend to exclude topics that occur infrequently, but found that these topics actually make up the majority of the data. In contrast, Bai et al. [9] research GitHub Event Data to recommend like-minded developers.

In their study, Borges et al. [10] investigated the factors that affect the popularity of GitHub repositories. They found a strong correlation between stars and forks, but weak correlations between stars and commits, as well as stars and contributors. They concluded that the programming language and application domain have a significant impact on popularity. In their study, Zhou et al [11] discovered 114,120 projects with 50 or more forks and 9,164 projects with 500 or more forks. They differentiated between social forks, which involve creating a personal copy, and hard forks, which involve branching off into a new development direction. Venigalla and Chimalakonda [12] analyzed 1.38 million artifacts from 950 GitHub repositories, revealing that pull-requests and issues contain substantial information useful for software documentation.

Distinct from other approaches, our method employs a list of keywords generated from topics already assigned to repositories. These encompass both popular and unpopular topics, organically created by users, negating the dependence on predefined dictionaries. We harness these keywords to extract features, implementing straightforward preprocessing techniques and word matching across various repository fields. This strategy eliminates the need for added knowledge, manual cleaning, or subjective weighting of keywords. Unlike other methods focused primarily on identifying topics for unlabeled repositories, our emphasis extends to pinpointing similar repositories. To this end, we deploy machine learning algorithms specialized in discerning repositories that align with our domain of interest.

3 Concept

Our approach aims to automatically cluster repositories from an open source sharing platform such as GitHub in such a way that users are quickly able to locate software from a specific domain or for a specific problem; in particular, this should be enabled by providing users with similar open source software projects for any given repository. To this end we propose to extract relevant features from the repositories, process them using methods from Natural Language Processing (NLP), and arrange the repositories into groups with appropriate clustering techniques.

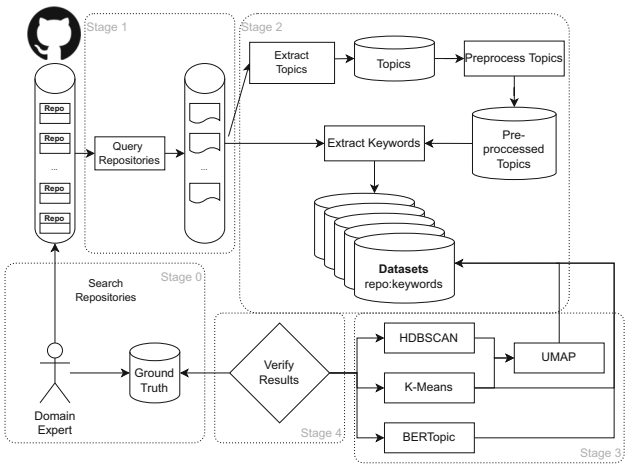

Fig. 1 Our approach

Figure 1 provides an overview of the steps involved in our approach. The procedure starts at the top left part of the figure, which represents a data base such as GitHub that contains a large number of open source repositories containing tools, configurations, best practices, IDS rules, hardening checklists, documentation, etc., for various domains. In a preparatory stage (stage 0) for evaluating our approach, a domain expert manually searches these repositories for a representative sample of projects with known domains; we assign these domains as labels to the repositories and consider this as the ground truth for evaluation. We point out that this stage is only required for evaluation and does not need to be conducted if the procedure is only used to identify similar repositories without validating the results.

Stage 1 of our approach involves querying the sharing platform with a set of pre-defined queries for a specific domain such as threat detection and store all gathered repositories in a data base for easy access. While it is theoretically possible to carry out the procedure on all available repositories, the sheer amount of data that needs to be handled renders this idea practically infeasible for large platforms such as GitHub. In stage 2 of our approach, topics that are provided by developers to indicate the problem domain of the software are extracted from the respective repositories. These topics are then processed using NLP techniques, including filtering, abstraction, weighting, etc. Our approach addresses the problem that topics are often missing, misleading, or ambiguous, by also leveraging auxiliary repository data such as the readme or about sections for descriptive keywords. Eventually, we obtain one or more datasets containing mappings between repositories and keywords.

Stage 3 of our approach centers around clustering of repositories based on their associated keywords. In this paper we conduct experiments with the well-known methods HDBSCAN and K-Means, as well as UMAP as an (optional) dimension reduction technique. In addition, we present our results using BERTopic [3], which is an approach that leverages neural networks to extract topics from a large corpus of documents. Either of these approaches yields a mapping of repositories to clusters; the idea is that these clusters contain similar repositories and that users may quickly find related projects for any given repository within the same cluster.

The final stage 4 of our approach is optional and only used for validating the results of the clustering procedure. Specifically, we locate the repositories from our ground truth in the cluster map and determine how many repositories with the same labels correctly end up in the same clusters, and whether clusters with different labels are correctly separated. This allows us to compute relevant metrics and compare the performance of different clustering algorithms with each other. Estimating the overall clustering accuracy from sampled data is necessary due to the fact that the whole corpus is too large to be labeled manually in its entirety; at the same time, methods such as BERTopic are not necessarily reliable and thus not suited to generate a ground truth. In the following sections, we discuss each stage of our algorithm in detail.

4 Data set

This section describes the dataset of open source software repositories used to develop and evaluate our approach. We first describe how we collected the dataset and then explain how we conducted feature extraction and labeling of individual repositories.

4.1 Collection

Clearly, data quality is a crucial factor for validating an approach to find relevant and similar projects within a large corpus of open source software. However, generating a representative dataset of open source projects in the security domain is a non-trivial task as it requires to select a subset of all available repositories and thus hinges on a very similar problem that the proposed approach attempts to resolve.

For our strategy to collect such a dataset we therefore first manually gather a list of 100 repositories in the threat detection domain by using Google search to locate GitHub repositories of intrusion detection software. For this purpose, we consulted our domain experts and gathered a list of known products in the open source incident detection domain such as "Suricata", "Snort", "Wazuh", "OSSec", "osquery", and "GRR". While searching for various tools on Google, we came across several awesome-lists such as “awesome-threat-detection”[7], which provides tools, GitHub projects, detection rules, etc., that we included into our initial list if they were open source repositories. In case that multiple awesome-lists contain links to the same repositories, we only included them once. Additionally to Google search we also used GitHub topics using keywords "ids", "siem", and "detection" to identify often stared repositories.

A basic search on GitHub for the keyword “security monitoring” only yields around 1,900 results; several of our manually selected 100 repositories from the security monitoring domain are not among the retrieved results. Our investigation of the reasons why these repositories are missing confirmed that only those repositories are retrieved where either the title, topics, or about section are exactly matching the keywords. By checking the missing repositories from our initial list we confirmed that they either do not have the required keywords in the respective fields or are missing the topics, the about section, or both entirely. Even when topics are provided in a repository, the lack of a common taxonomy of topic keywords make it unlikely to obtain all relevant repositories. For example, we found that at least the topics “threatintel”, “threat-intelligence”, and “ti” all relate to the same concepts, but the set of all related repositories cannot be retrieved without exactly specifying all possible variations of the keyword *threat intelligence*.

Aware of these issues, we noticed that most repositories contain a readme section with a detailed description of the repository and including the necessary keywords to be retrieved by the query. We therefore extend the GitHub search query to also consider the readme section (this is achieved by entering “in:readme, topics, about, title” in the search bar) and obtained around 106,000 repositories. Analyzing these results showed that this time a higher fraction of our initially selected repositories was retrieved; however, at the same time, the number of false positives, i.e., repositories that contain the keywords “security” and “monitoring” in their readme section but are in fact not relevant for the security domain, increased as well. While false positives are generally not desirable, we consider this situation ideal for the purpose of our evaluation as our proposed approach should not only identify relevant and similar repositories within a set of security-related software but instead also be able to differentiate them from software related to other domains, which requires a diverse set of projects. While we did not manually label all false positives, we intentionally included the broader set of repositories to reflect a realistic and noisy search scenario. This setup allows our clustering approach to demonstrate its effectiveness in identifying meaningful groups of relevant repositories within a diverse and partially unrelated dataset. In the end, we assembled 24 search queries such as “ids”, “ips”, “security monitoring”, “nsm”, and oth-

[7] https://github.com/0x4D31/awesome-threat-detection

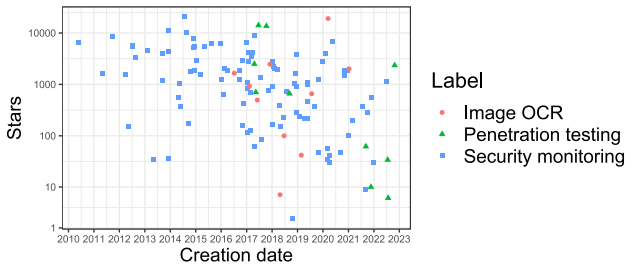

Fig. 2 Creation date and stars of ground truth repositories

ers, that allowed us to retrieve all of our 100 initially selected repositories among a total of 221,971 repositories obtained from the search.[8] Note that we limit the search space and filter out dead forks and inactive repositories by requiring that retrieved repositories have at least two stars, one fork, and two followers. Our main intention for this is to omit "social forks", i.e., public copys of repositories usually comprising minor changes such as feature implementations, that would introduce a large number of de-facto duplicates into our dataset [11].

4.2 Ground truth

To evaluate the effectiveness of our clustering approach, we created a manually labeled ground truth set consisting of 120 repositories from three distinct application areas: 100 from the domain of threat detection (security monitoring) as described in Sect. 4.1 and 10 each from the penetration testing and image Optical Character Recognition (OCR) domains. These sets were selected based on expert knowledge and serve as representative examples to validate whether our approach can distinguish repositories with substantially different functionality and topical focus.

A ground truth that specifies which repositories belong to the same domain or problem space is essential for quantitative evaluation, as GitHub-provided metadata such as topics are often too inconsistent or noisy to serve this role. Given the scale of our dataset (221,971 repositories), fully annotating all entries was infeasible. We therefore opted to manually label a smaller, curated set of 120 repositories. Each repository was assigned one of three coarse-grained labels: *security monitoring*, *penetration testing*, or *image OCR*.

We use the resulting labeled set to validate whether our clustering approach can group repositories from similar domains while separating topically distinct projects. Figure 2 shows the distribution of stars and creation dates for the ground truth repositories. Interestingly, many selected repositories have a higher star count than average, which we attribute to their frequent inclusion in curated lists such as awesome-lists-further highlighting their relevance and visibility in practice.

To minimize selection bias during the extraction of the penetration testing and OCR repositories, two domain experts searched the full dataset using domain-specific keyword queries (e.g., "penetration testing", "red teaming", "Tesseract", "OCR"), targeting the same fields used during collection (e.g., topics, README, description). From the resulting candidate sets, they randomly sampled 10 repositories per domain, excluding projects that lacked sufficient documentation to allow confident labeling.

The threat detection repositories span a diverse range of project types, including detection rule sets, frameworks, and standalone tools. In contrast, the penetration testing and OCR sets are more homogeneous, mostly comprising standalone tools, lists and documentation. Notably, while OCR is conceptually distinct from security, overlaps in technical characteristics (e.g., use of CLI tools, Python code, or shared keywords like "detection") make separation non-trivial. The penetration testing set poses a different challenge, as it belongs to the broader security domain and shares significant topical overlap with threat detection, making differentiation more difficult. This setup allows us to test whether clustering can capture both broad inter-domain and subtle intra-domain differences.

We explicitly acknowledge that this manual labeling process introduces potential bias. Repository selection, while guided by consistent criteria and domain expertise, remains inherently subjective. This subjectivity may affect evaluation results and limits generalizability. Even for domain experts, assigning a single definitive label to a repository can be challenging, especially when software fulfills multiple functions or lacks clear documentation. For example, whether the `osquery` project falls under detection or response is debatable, and classifications may vary depending on the schema used.

Alternative classification approaches, such as those proposed by Zanartu et al. [13], categorize repositories by type (e.g., software, documentation, web libraries). Others, like Mahn et al. [14], suggest mapping to NIST functions. These more granular or type-specific schemes can offer value but also increase complexity and disagreement in label assignment. To reduce such ambiguity, we chose coarse-grained domain labels, which we consider to be the most objective and consistent for our evaluation goals.

4.3 Feature extraction

From our dataset comprising 221,971 repositories we extracted the following features: repository name, about section, associated topics, readme section, root directory tree, programming languages (which is automatically estimated by GitHub based on the files stored in the repository), size, whether the

[8] We refer to our GitHub repository for the full list of repositories: https://github.com/ghml23/ghml23

repository is archived or a fork, whether the repository has a Wiki or a download section, the number of open issues, the dates of creation and last update, and the number of stars, forks, commits, watchers, releases, and contributors. Most of these attributes are relatively straightforward to process as they are categorical values or quantities, except for the about and readme sections that are written by developers and designed to be human readable rather than machine readable.

Specifically, the readme section is either a markdown-styled file or plain text and contains highly unstructured information. For example, rather than explaining the application context or the problems solved by the project, many readme sections mainly provide details on how to install and build the software (e.g., snort3[9]), contain links to the installation or user guide and focus on project contributions (e.g., suricata[10]), or provide hardly any textual information at all (e.g., Bearded-Avenger[11] or Securityonion[12]). Despite the diversity and high complexity of the readme section, we argue that the provided information in that file is highly relevant for project discovery as outlined in Sect. 4.1 and thus propose to apply advanced machine learning techniques on that data. We will go into detail on this matter in Sect. 5.

5 Approach

The goal of our approach is to be able to group and sort open source software repositories from the domain of threat detection by their similarities. To this end, we need to pre-process the features described in Sect. 4.3 in such a way that they are suitable to be represented in a data structure that enables similarity-based comparison. We achieve this by applying techniques from natural language processing (NLP), dimension reduction, and clustering algorithms. The ground truth dataset with 120 repositories is used separately for evaluation purposes and not part of our approach.

In the remainder of the paper, we use the term keyword to refer to relevant textual features (e.g., topic labels, readme tokens) extracted from GitHub repositories. When these keywords are vectorized for machine processing, we refer to them as terms in the feature matrix.

5.1 Pre-processing

This section provides some insights into the dataset using methods from natural language processing (NLP). Based on the findings, we then outline our strategies to pre-process the dataset and derive several evaluation datasets.

Analysis of the dataset. Despite the facts that topics are often missing (only 31% of the repositories in our dataset use topics) and have no common taxonomy as stated in the previous sections, they still provide a source of information that can be useful to assess the similarities of repositories. Investigating the topics in our dataset shows that the most common topics are python, deep-learning, machine-learning, computer-vision and javascript. In some cases there is also a version number appended to words that represent technologies, such as python3, which is also within the top 20 of most used topics.

As a first step we tokenize topics comprising multiple words. However, manually investigating the results quickly showed that separating composite words such as “threat-detection” into “threat” and “detection” is not desirable as the combination of these words is a better indicator for the purpose of the repository than the separated words. One approach to deal with this situation is the use of bigrams [15], however, we refrain from this idea as it introduces high computational complexity in a large dataset. Instead we preserve the information of composite words by combining them into one word and removing the hyphen, e.g., “machine-learning” is transformed into “machinelearning”.

State-of-the-art NLP techniques also usually remove any words below a given character count. Given that some of our topics are made up only of two or even a single character (e.g., the programming languages “C” and “R”), we omit this step. Another common pre-processing technique that is tricky to apply in our situation is lemmatization, which aims to merge different words with the same or very similar meaning into a single one, thereby improving comparability. Our experiments showed that the ntlk[13] lemmatizer used with stemming replaces frequent and highly relevant topics such as “ids” and “ips”, which are abbreviations for intrusion detection/prevention system, with “id” and “ip”, thus incorrectly changing their meaning. We, therefore, proceed without stemming. Other than tokenizing, we do not modify or remove any topics retrieved from the dataset, including fixing spelling errors, validating topics for certain repositories, or filtering outliers. In total 93,055 unique topics were used, while 58,849 (63.24%) of them are only used in one repository.

Other than topics, around 99% of repositories include readme sections, making them an essential source of information to determine the purpose of repositories. After a review of our initial set of repositories, it is apparent that many readme sections are written in markdown language and contain HTML code, along with code examples. Additionally, some sections may involve languages other than English. We filter out all markdown code, HTML code, and web links to pre-process the readme sections. We then check the first 200

[9] https://github.com/snort3/snort3

[10] https://github.com/OISF/suricata

[11] https://github.com/csirtgadgets/bearded-avenger

[12] https://github.com/Security-Onion-Solutions/securityonion

[13] https://www.nltk.org/_modules/nltk/stem/wordnet.html

words of the readme section against an English dictionary to validate their language. The results show that around 95% of all repositories included at least 50% of words that are either English or topics assigned to that repository. Carrying out the same procedure for the about section shows that around 88% of all repositories have an about section, with almost all of them (in total 86%) written in English or coinciding with the topics. These results confirm that English is the dominant language across repository metadata, supporting our focus on English keywords and the use of English-language during preprocessing.

Generation of evaluation datasets. To investigate the influence of the various aforementioned pre-processing strategies applied on the data, we create nine separate keyword lists from the original dataset comprising 221,971 repositories.

Table 1 summarizes the keyword lists leveraging only topics of repositories. Note that repositories without any topics are excluded, which means that some of the 120 labeled repositories that make up our ground truth are missing. As outlined in the table, for keyword list we apply a combination of pre-processing strategies including tokenization (T), lemmatization (L), and removal of hyphens in compound words (C). Except for D_T1, which remains entirely unchanged, we also eliminate so-called stopwords during tokenization (T) with little relevance to any specific topic, for example, words such as "I", "there", "and", etc.. Importantly, hyphen removal is applied before tokenization. This step replaces hyphenated compound words (e.g., "network-based") with merged forms (e.g., "networkbased"), which are then preserved as single tokens. As a result, applying C can significantly increase the number of distinct keywords, as it introduces many new variants not otherwise tokenized. This explains why D_T4, which applies both C and T, contains more than twice the number of keywords compared to D_T3, which only applies T.

Moreover, we apply a filtering strategy that neglects infrequent topics. The main idea is that a reduced set of keywords results in smaller datasets and therefore less computation required. Further we argue that topics that occur in several repositories are better suited for grouping them than those that occur only in few or even only a single repository. We therefore generate D_T6 to only include repositories with common topics that are present in at least r_{min} (minimum repositories) of repositories, where we select $r_{min} = 34$ (0.005% of repositories with topics) for our experiments.

We further use the list of pre-processed topics derived from all repositories as keywords, to determine whether any keywords occur in the readme, about, tree, and name sections of the repositories. Note that the readme section has been cleaned as previously described Sect. 5.1 by filtering markdown and html code.

Table 1 Keyword lists created from topics

Name	C	T	L	r_{min}	#Keywords
D_T1				0	93,053
D_T2		✓	✓	0	43,131
D_T3		✓		0	44,924
D_T4	✓	✓		0	89,406
D_T5	✓	✓	✓	0	88,364
D_T6	✓	✓	✓	34	1,567
D_C1		✓		4	9,652
D_C2	✓	✓	✓	4	14,653
D_C3	✓	✓	✓	69	773

Since the total size of a dataset largely increases after incorporating the readme section as a source of keywords, it is necessary to leverage r_{min} to reduce the number of keywords and eliminating irrelevant ones that are not shared among sufficiently many repositories. In Tab. 1 we list the number of keywords matched against the input fields, while $r_{min} = 69$ (0.01%) yields only 773 keywords. The values for r_{min} were empirically determined based on the observed topic frequency distribution, in order to retain sufficiently common topics while excluding sparse or noisy ones.

To clarify the construction of our evaluation datasets, we distinguish between two types: **D_T**x and **D_C**x. The **D_T**x datasets are filtered based solely on GitHub *topics*. Specifically, we apply a minimum repository count threshold r_{min} to retain only those topics that appear in at least r_{min} repositories, under the assumption that frequently used topics are more reliable for grouping. In contrast, the **D_C**x datasets are based on *content-derived keywords*, which are extracted from multiple metadata sources (e.g., README files, repository titles, and directory trees). These keywords are vectorized using `CountVectorizer`, and a minimum document frequency threshold df_{min} is applied to reduce dimensionality and remove infrequent terms.

5.2 Dimension reduction

The evaluation datasets described in the previous section comprise unstructured data, specifically, lists of topics and words derived from the various sections. While these words are semantically expressive, they are intended for human rather than machine consumption. We therefore need to transform the unstructured data into a format that is easier to process by clustering algorithms. To this end we use CountVectorizer[14] to generate a term frequency matrix for all words in all sections. Thereby, we select a minimum

[14] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

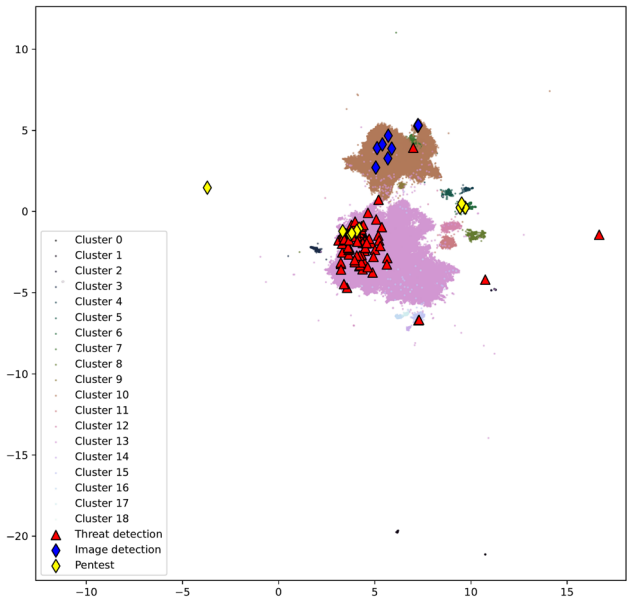

Fig. 3 UMAP HDBSCAN Cluster labels on dataset of extracted D_C2 keywords on readme

document frequency (df_{min}) of four to omit those keywords that only occur in few repositories and are therefore deemed irrelevant, which reduces the number of features to a more manageable size for further calculations. We chose $df_{min} = 4$ to avoid instabilities and crashes due to memory limits on our laboratory machine. In the resulting term frequency matrix, each row represents a repository, each column represents a distinct keyword, and each cell stores the number of times a specific keyword appears in the respective repository. For example, the matrix generated for the D_C2 dataset is 221,971 by 14,653 in size.

To reduce the dimensions of this matrix, we use Uniform Manifold Approximation and Projection (UMAP), which captures complex nonlinear structures present in high-dimensional data and projects them onto a lower-dimensional space. UMAP excels at preserving both global and local structures within the data, striking a balance between maintaining neighborhood relationships and capturing broader patterns [16] compared to traditional techniques such as T-distributed Stochastic Neighbor Embedding (T-SNE) and Principal Component Analysis (PCA). When estimating manifold structures of data, we use a 2-dimensional representation with a minimum distance of zero (meaning that the repos are as close as possible) and 30 $n_{neighbors}$ to examine larger neighborhoods. Overall, this strategy allows us to harness the advantages of UMAP's ability to preserve intricate structures while accommodating the nuanced nature of GitHub repository data.

5.3 Clustering using HDBSCAN and K-means

Our experiments to group repositories by similarity revolve around two different clustering algorithms: Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [17] and K-Means [18]. HDBSCAN is a strong density-based clustering algorithm that can uncover clusters of various shapes and sizes in the data. It automatically identifies the number of clusters and detects noise points, making it the ideal choice for identifying intrinsic patterns in our GitHub repository datasets. We applied HDBSCAN to the transformed data and obtained clusters based on local density variations. These clusters helped us identify repositories that share similar characteristics and discover cohesive groups within the dataset.

To explore the structure of the data further, we also apply K-Means, a popular partitioning algorithm, on the transformed dataset. We additionally apply K-Means directly to the original data without UMAP's dimension reduction to investigate the impact of dimension reduction on clustering outcomes. In Sect. 6.1, we compare the results obtained from HDBSCAN and K-Means (with and without the UMAP-transformation) to evaluate the quality and coherence of the obtained clusters, the distribution of repositories across clusters, and the interpretability of the resulting groups.

5.4 Topic extraction with BERTopic

We further apply BERTopic [3] algorithm in its default configuration to analyze two datasets: (i) topics section according to D_T2, (ii) readme section using keyword list D_C2. We leverage BERTopic to extract and uncover underlying topics within the dataset, shedding light on latent patterns and themes. We compared the results of BERTopic with our manual approach, which involves extracting and labeling topics based on domain expertise, to determine its effectiveness. The purpose of this assessment is to demonstrate the algorithm's capacity to independently distinguish significant clusters and topics that follow patterns identified by humans. We present the results in Sect. 6.2.

6 Comparative analysis and insights

This section presents the results of our applied clustering and topic extraction algorithms.

6.1 Clustering using HDBSCAN and K-means

In this section we compare the resulting clusters generated by HDBSCAN and K-Means, and assess the influence of UMAP on the K-Means clustering. This comparison enables a better understanding of the GitHub repository landscape,

Table 2 Clustering results

Algorithm	T	A	R	N	D	L	Clusters	Keywords	F1	PPV	TPR	L. Tot. in %	L. GT in %	A. Tot. in %	A. GT in %
UMAP + HDB	x	x	x				5	D_C2	0.93	0.90	0.96	98.36	100.00	4.99	2.50
UMAP + HDB	x	x	x	x	x		13	D_C1	0.93	0.90	0.95	98.38	100.00	5.54	2.50
UMAP + HDB		x	x	x			10	D_C2	0.92	0.89	0.96	98.36	100.00	2.69	1.67
UMAP + HDB			x				18	D_C2	0.92	0.90	0.93	98.36	100.00	8.18	5.00
UMAP + HDB	x	x	x	x	x		11	D_C2	0.92	0.85	1.00	98.36	100.00	4.68	0.83
UMAP + HDB	x	x	x	x			25	D_C1	0.92	0.90	0.93	98.38	100.00	5.95	5.00
UMAP + HDB	x	x	x	x			22	D_C2	0.92	0.90	0.93	98.36	100.00	8.51	7.50
UMAP + HDB		x	x				33	D_C2	0.91	0.89	0.94	98.36	100.00	13.55	19.17
UMAP + HDB	x		x				10	D_C3	0.91	0.84	0.98	98.18	100.00	2.43	1.67
UMAP + HDB	x	x	x				21	D_C1	0.91	0.89	0.93	98.38	100.00	14.03	14.17
UMAP + HDB			x				33	D_C1	0.90	0.84	0.96	98.38	100.00	7.41	2.50
UMAP + HDB		x	x	x	x	x	64	D_C3	0.89	0.84	0.95	98.18	100.00	16.44	5.00
UMAP + HDB	x						8	D_T2	0.84	0.79	0.89	31.45	81.67	14.41	9.18
UMAP + HDB				x		x	7	D_C2	0.83	0.92	0.76	12.79	32.50	42.98	15.38
UMAP + HDB	x						15	D_T6	0.81	0.78	0.84	28.52	81.67	35.51	11.22
UMAP + HDB			x				20	D_C3	0.80	0.86	0.75	98.18	100.00	14.99	13.33
UMAP + HDB	x						15	D_C3	0.78	0.78	0.77	27.19	80.00	28.99	8.33
UMAP + K-Means	x						10	D_T3	0.77	0.78	0.77	31.07	81.67	0.00	0.00
UMAP + K-Means	x						10	D_T4	0.76	0.78	0.74	29.89	81.67	0.00	0.00
UMAP + K-Means	x						16	D_T6	0.75	0.79	0.71	28.18	81.67	0.00	0.00
UMAP + HDB	x	x	x	x			37	D_C3	0.73	0.87	0.62	98.18	100.00	22.74	38.33
UMAP + K-Means	x						10	D_C2	0.72	0.77	0.68	29.28	81.67	0.00	0.00
UMAP + K-Means	x						10	D_T1	0.72	0.77	0.67	31.08	81.67	0.00	0.00
UMAP + HDB		x	x	x	x		62	D_C3	0.72	0.80	0.65	98.18	100.00	40.32	20.83
UMAP + K-Means	x						10	D_C3	0.71	0.76	0.67	27.07	80.00	0.00	0.00
UMAP + K-Means	x						10	D_T2	0.70	0.77	0.64	31.09	81.67	0.00	0.00
UMAP + K-Means	x	x	x	x			10	D_C3	0.68	0.85	0.57	98.17	100.00	0.00	0.00
UMAP + K-Means					x		10	D_C2	0.67	0.80	0.57	99.86	100.00	0.00	0.00
UMAP + K-Means		x	x	x			13	D_C2	0.65	0.88	0.51	98.35	100.00	0.00	0.00
UMAP + K-Means	x	x	x				13	D_C1	0.64	0.86	0.51	98.38	100.00	0.00	0.00
UMAP + K-Means	x	x	x				13	D_C3	0.64	0.80	0.53	98.17	100.00	0.00	0.00
UMAP + K-Means		x	x				16	D_C2	0.63	0.81	0.52	98.36	100.00	0.00	0.00
K-Means		x					10	D_C2	0.63	0.74	0.55	85.01	96.67	0.00	0.00
UMAP + K-Means	x	x	x	x			13	D_C1	0.63	0.81	0.52	98.38	100.00	0.00	0.00
UMAP + K-Means	x		x				10	D_C3	0.62	0.79	0.51	98.17	100.00	0.00	0.00
K-Means		x					13	D_C3	0.58	0.70	0.50	75.94	84.17	0.00	0.00

in particular, allows to investigate whether software projects are relatively distinct and form many diverse and easily discernible clusters of few or even single repositories, or whether few comparatively large clusters exist that correspond to specific types of repositories, such as code, documentation, awesome-lists, etc. The goal of this investigation is to be able to determine whether the resulting clusters support users in exploring similar repositories and subsequently enabling more informed decision-making for software discovery and recommendation.

For objective and quantitative evaluation of these tasks we compute precision, recall, and F1 score based on our ground truth of 120 labeled repositories. As outlined in Sect. 4.2, our labels differentiate between repositories from the security monitoring and penetration testing domain as well as repositories related to image OCR. We consider a clustering as good if many of the repositories with the same labels end up in the same clusters, while at the same time repositories with different labels should not end up in the same clusters. Note that the actual number of clusters generated for the data is only of little relevance for the use-case of finding similar repositories, and anyway cannot be known beforehand. Given these requirements, we opt for the idea to count all possible pairs of repositories as described by Schütze et al. [19]. In particular, this means that a pair of identically labeled repositories in the same cluster counts as a true positive (TP), a pair of identically labeled repositories in different clusters counts as false negative (FN), a pair of differently labeled repositories in different clusters counts as true negative (TN), and a pair of differently labeled repositories in the same cluster counts as false positive (FP). Based on these measures, we compute precision or positive predictive value $PPV = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$, the recall or true positive rate $TPR = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$, and the $F1 = \frac{2 \cdot \mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV+TPR}}$.

Table 2 displays the outcomes of applying the various keyword lists described in Tab. 1 on various sections of a GitHub repository. In particular we examined sections topics (T), about (A), readme (R), name (N), description (D) and languages (L). We only list the best result (F1 score) of each run (different cluster sizes) on a particular dataset when it exceeds 0.5 true positive rate and 0.5 true negative rate. We sort the results depending on the F1 score. Next to precision, recall, and F1 score, the table states the total number of clusters as well as the number outliers (A.) in percent, i.e., repositories that are identified as too dissimilar and therefore not assigned to any clusters, from the entire dataset as well as the ground truth (GT).

We also provide the percentage of repositories with assigned cluster labels (L.) in both the entire dataset and the ground truth. For instance, the first line shows that using the D_C2 method and keyword list on the topic, about, and readme fields results in 98.36% of repositories being assigned a cluster label, and 100% within our ground truth. This is a stark contrast to the best-rated result utilizing D_T2, applied only to the topic field, which labels a mere 31.45% of the dataset, as fewer than a third of repositories have any topics assigned. The high labeling percentage using D_C2 can be attributed to the fact that nearly all repositories have a readme containing at least one of the extracted keywords. In the absence of such keywords, there is no information available for assigning a cluster label.

It is crucial to highlight that the accuracy in Tab. 2 is not adversely affected when a repository is unlabeled or identified as an anomaly. Consequently, our keyword extraction and matching strategy demonstrates improved clustering performance in our experiments, benefiting from the additional features extracted from various metadata fields. While these results indicate potential for enhancing repository discovery, further validation through user studies and expert feedback is needed to assess real-world utility.

When applied on a large dataset with many features, UMAP in combination with HDBSCAN outperforms all other methods and delivers almost perfect results. Our approach achieves a recall of 96% and precision of 90%, with only 2.5% anomalies.

This means that there are none or hardly any repositories that are incorrectly grouped in the same cluster, indicating that the approach was able to successfully separate security monitoring software from repositories dealing with penetration testing and image detection. An important observation that can be made from this table is that clustering based only on topics and not considering additional sources of information such as readme and about sections yields comparatively poor results, e.g., UMAP combined with HDBSCAN achieves the best clustering accuracy of all topic-based datasets on D_T2 but yields only an F1-score of 84%. Another insight is that K-Means is generally unable to compete with HDBSCAN as the highest achieved F1-score is only 77% when applied on D_T3. It is important to note that while K-Means yields higher F1-scores on D_T3, D_T4, and D_T6 on just the topics, the omission of repositories without topics from these datasets, evident from the lower number of labeled repositories, means that several relevant repositories cannot be retrieved at all. This limitation is not reflected in the precision, recall, and F1-score metrics, which only consider the available (i.e., topic-annotated) repositories. Accordingly, we determine that while K-Means is unable to yield good results on the topic-based datasets as well as the datasets with combined attributes, good results can be achieved when using HDBSCAN on complex datasets containing information from readmes and about sections in addition to topics.

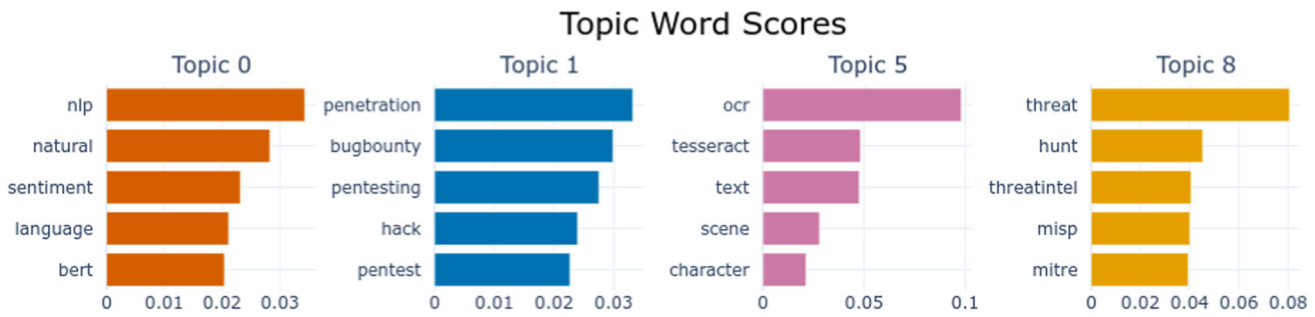

Fig. 4 Example of BERTopic extracted topic clusters

6.2 Topic extraction with BERTopic

Note that HDBSCAN assigns all repositories that deviate too much from the overall distribution in a separate outlier cluster. K-Means on the other hand assigns all repositories to clusters; however, UMAP might disconnect some of the vertices resulting in isolated repositories that cannot be clustered and are thus also considered as outliers. This is why UMAP in combination with K-Means shows lower "L. Total" and "L. GT" values even in D_C1-6 where repositories are not omitted due to their lack of topics. Note that only those repositories of the ground truth that were not identified as anomalies are considered in the calculation of our evaluation metrics. The reason for this is that these repositories are not clearly associated to any class and thus difficult to integrate in the evaluation metrics.

Manually analyzing the results showed that some repositories with the same label form separate clusters that are specific to certain problems, e.g., configuration files of security solutions likely have inherently different repository structures or contents than SIEM systems. This may lower precision and recall when very fine-grained labels are used, but should not affect our evaluation that relies on more coarse-grained labels.

Figure 3 shows the resulting cluster maps of HDBSCAN with UMAP applied on readme using D_C2, which are projected on a two-dimensional plane, where each point represents a repository and is colored according to the associated cluster. We also visualize the repositories from the ground truth in the plots, where triangle shapes represent repositories corresponding to the threat detection label, yellow diamond shapes highlight penetration testing repositories and blue diamond shapes correspond to repositories in the image OCR domain.

BERTopic allows us to identify clusters that are more complex than the ones produced by the clustering mechanisms from the previous sections, thereby providing a more elaborate view on the underlying data patterns. Specifically, BERTopic provides a list of topics and their respective relevance for a topic cluster. Figure 4 illustrates four exemplary sample clusters.

Since topics identified by BERTopic are more specific than the labels we used to categorize repositories, we found that there is only a low correspondence between these two schemes. For example, BERTopic detected 709 topic clusters in D_T2, while our clustering approaches yield only 8-13 clusters (cf. Table 2). As a consequence, applying our validation method on the BERTopic clusters resulted in a poor F1-score of 0.19, with perfect precision of 1 but a low recall of only 0.01.

When running BERTopic in default configuration HDBSCAN is utilized, which results in 25,299 outliers in the entire dataset and 33 in our ground truth (D_T1 on topics). However, as indicated in Fig. 4, topic cluster 5 primarily corresponds to repositories related to image OCR; eight out of the ten image OCR repositories from out ground truth ended up in this cluster, while the two remaining ones were not assigned to any clusters and thus count as outliers. Moreover, 13 repositories labeled as threat detection repositories according to our ground truth correctly end up in the threat intelligence/threat hunting topic cluster 8. Topic 1 corresponds to the penetration testing cluster with 5 repositories correctly assigned to this cluster. When executed on the readme dataset extracted with D_C2, we identified 63 outliers in our ground truth and 33 clusters, each containing only one to two repositories. The runtime escalates, increasing sevenfold to 121 minutes. These outcomes are impractical for our needs.

7 Discussion

Our objective was to identify repositories in the threat detection domain, prompting us to turn to GitHub, the world's largest open-source software code base. However, this journey was not without significant challenges. Initially, we were confronted with the stark limitation of finding similar repositories or those belonging to a specific domain. Utilizing common search queries in this domain often yielded

an overwhelming number of repositories, underscoring the inadequacy of the search functionality. Although GitHub has introduced the "topics" feature, allowing creators to classify their repositories, this did not entirely mitigate the issue. Our attempt to leverage topics for clustering similar repositories led us to create a dataset comprising 221,971 repositories, curated through approximately 22 search queries with keywords commonly associated with security monitoring and threat detection.

Yet, a notable observation was that only a third of these repositories had topics assigned. Moreover, there is an inconsistency in the topics even among similar repositories. For example, the "Suricata"[15] repository lists the keywords "ids", "ips", "nsm", "threat-hunting", and "intrusion-prevention" system as topics. At the same time, the about section describes Suricata in the written form of "nsm" (network security monitoring) and does not mention any of the abbreviations "ids", "ips", or "nsm". Also, the keywords threat or hunting only occur as a topic but not in readme or about sections. These findings indicate that abbreviations and synonyms as well as a non-uniform taxonomy make it difficult to find similar software. To navigate this challenge, we devised a method to glean information from other repository artifacts. We utilized an extracted set of keywords, rooted in the topics already assigned within our dataset. This strategy proved successful, enabling us to extract features from 98.36% of all repositories in the dataset.

In our experiments, we were able to seperate threat detection repositories from image OCR (Optical Character Recognition) and penetration testing repositories. Remarkably, our method yielded results with a higher accuracy than could be achieved by relying solely on the topics assigned by the repository creators. For example, upon manually verifying the cluster labels derived from the extracted features of the readme section using D_C2, it became apparent that the separation of penetration testing repositories into a distinct cluster was more effective (up to 50%) compared to using only topics and D_T1-6. The most detailed results, with 64 clusters, were obtained by employing the smallest keyword list across various sections including about, readme, name, tree, and language still achieving a 0.89 F1 score but with an increased number of anomalies within the ground truth of 16.44%. Yet when applied to topics and readme D_C3 still achieves very good results.

This is encouraging, as the dataset extracted using D_C3 on the readme is 3.4 times smaller compared to that extracted using D_C2. Cluster runs vary in duration, taking anywhere between 1-9 seconds depending on the cluster size, for HDBSCAN or K-Means when executed on UMAP embedding, which itself takes approximately 600 seconds (D_C3). This is in contrast to the 5-10 seconds for HDBSCAN, 1-3 seconds for K-Means, and 850 seconds for UMAP alone (D_C2). All tests were conducted on hardware equipped with an 8-core i9-9900k and 32GB RAM. It is notable that K-Means, when executed without UMAP, is markedly impacted by the cluster/dataset size, requiring up to 950 seconds per run. One run of BERTopic on readme using D_C2 takes 7200 seconds, which is more than 8 times longer than HDBSCAN + UMAP.

These runtime comparisons indicate that our approach can be scaled to larger repository sets while remaining computationally efficient. Compared to transformer-based methods such as BERTopic, which required up to 121 minutes on the same dataset, the combined use of UMAP with HDBSCAN or K-Means offers improvements in runtime while still producing meaningful clusterings. Nonetheless, future work could incorporate a more systematic performance evaluation, including memory profiling and scalability testing. The poor runtime and clustering behavior of BERTopic are likely due to the combination of long inference times for transformer-based embeddings and the noisy nature of README texts. Future work could investigate lighter or domain-adapted embedding models, pre-clustering techniques, or more aggressive topic merging strategies to improve BERTopic's usability in large-scale code repository analysis.

Our evaluation confirms that our approach of using popular topics to extract keywords that can be used for finding similar repositories is promising. Consider, for example, "snort3" and "Suricata". Both repositories contain actively maintained open source software and are de-facto standard software for network intrusion. The only difference is that the "snort3" repository does not have any topics assigned by the authors, nor does it provide an about section. Therefore it cannot be found with the default search using GitHub, which does not search within readme sections. It is therefore necessary to apply strategies for topic extraction as outlined in this paper as well as similarity functions such as Jaccard ($J(A, B) = \frac{|A \cap B|}{|A \cup B|}$) on the sets of extracted keywords A and B of both repositories respectively.

We tested the Jaccard similarity on the aforementioned repositories from different versions of our datasets and found that the achieved scores differ depending on the preprocessing methods applied on the data. For example, in D_C2 the repositories yield a Jaccard similarity of only 0.2, while in D_C3, which contains a more restricted set of common keywords that appear in a minimum amount of 69 repositories, they yield a similarity of 0.31. This effect can be explained by the fact that Jaccard similarity is known to suffer from limitations when analyzing uneven, lengthy, and unstructured text files. As a consequence, longer texts may exhibit inflated similarity scores due to the sheer number of unique keywords. Nonetheless, Jaccard similarity allows to find highly similar repositories. When calculating Jaccard to every other repository in the dataset utilizing D_C3

[15] https://github.com/OISF/suricata

we find the following three repositories as most similar to snort3: Autosnort3[16] (0.39), snort3_extra[17] (0.375), CodeExecutionOnWindows[18] (0.372). While the first two results are excellent matches, CodeExecutionOnWindows focuses on attacking rather than detection, but share some keywords (shell, detection, log, ...). This is just one example of the difficulty of separating penetration testing from threat detection repositories.

Future work can involve refining natural language processing models to distill a clearer representation of a repository's objectives and problem-solving capabilities. The results of BERTopic are promising, and further refinements can be made in the future. Using its default configuration did not meet our expectations due to an enormous number of outliers and overly fine clustering, resulting in the largest cluster in our ground truth comprising only 2 repositories. To achieve results akin to our requirements, careful tuning and additional research are necessary. We expect that the development of comprehensive synonym databases and clear taxonomies could significantly refine the precision of our topic recommendation framework. Furthermore, incorporating expansive language models like GPT-3.5 [20] might enhance the accuracy and depth of our topic extraction even further. Our approach suggests several potential enhancements. Strategies such as stopword extraction, topic weighting, topic cleaning, and spell-checking can improve the precision and relevance of our topic recommendations.

Evaluating repository quality is vital. Though stars and forks offer insights, their potential bias and vulnerability risks underscore the need for a comprehensive assessment framework beyond conventional metrics.

Though various GitHub topic recommendation approaches exist (cf. Sect. 2), none are implemented on the platform. Our paper introduces a method enabling users to find similar repositories, even without pre-defined topics or keywords. Initially aimed at distinguishing security monitoring repositories, the approach shows promising signs of generalizability across domains. Our evaluation considered the impact of parameters like similarity thresholds on clustering. Higher similarity settings offered finer clustering.

Crawling GitHub is time-consuming, and with the growing number of public repositories, a more efficient search and recommendation engine is needed. Current API limitations hinder the extraction of comprehensive user contribution data, yet such information could enhance repository similarity measures and improve recommendations.

[16] https://github.com/da667/Autosnort3

[17] https://github.com/snort3/snort3_extra

[18] https://github.com/pwndizzle/CodeExecutionOnWindows

8 Conclusion

In this paper we present an approach that is highly effective in separating security monitoring from penetration testing and image OCR repositories on GitHub. By leveraging a set of commonly used topics to extract keywords from various sections of each repository, including the title, about, readme, tree, and topics, we are able to accurately cluster repositories using UMAP and HDBSCAN, even though 68% of all repositories do not provide any topics. For our evaluation, we labeled a set of 120 repositories though domain knowledge and are able to achieve an F1-score of 0.93, precision of 0.90, and recall of 0.96 on that sample. Our experiments suggest that our approach provides a reliable and efficient method to obtain similar repositories for any given repository.

A current limitation of our approach is its reliance on keyword-based features, which may not capture deeper semantic similarities between repositories. In future work, we plan to explore the use of embedding-based representations and pre-trained NLP models to improve semantic understanding and clustering quality. In future versions of our topic clustering approach we aim to complement our technical evaluation with user-centric validation, such as usability studies or feedback from domain experts, to better assess the practical usefulness of the clustering results in real-world scenarios.

Funding Open access funding provided by AIT Austrian Institute of Technology GmbH. This work was partly funded by the Austrian Research Promotion Agency (FFG) project SPOTTED (4148328).

Data Availability Our results, approach and the dataset are published open-source and are available at https://github.com/ghml23/ghml23.

Declarations

Financial Support and Administrative Support Financial support, administrative support, and article publishing charges were provided by Austrian Research Promotion Agency.

Competing interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Dhir, S., Dhir, S.: Adoption of open-source software versus proprietary software: An exploratory study. Strategic Change **26**(4), 363 (2017)
2. Widyasari, R., Zhao, Z., Cong, T.L., Kang, H.J., Lo, D.: Topic Recommendation for GitHub Repositories : How Far Can Extreme Multi-Label Learning Go ?, 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) pp. 167–178 (2023). https://doi.org/10.1109/SANER56733.2023.00025
3. Grootendorst, M.: Bertopic: Neural topic modeling with a class-based tf-idf procedure, arXiv preprint arXiv:2203.05794 (2022). https://arxiv.org/abs/2203.05794. Accessed: 2025-04-05
4. Rocco, J.D., Ruscio, D.D., Sipio, C.D., Nguyen, P., Rubei, R.: TopFilter: An approach to recommend relevant GitHub topics. International Symposium on Empirical Software Engineering and Measurement (2020). https://doi.org/10.1145/3382494.3410690
5. Di Sipio, C., Rubei, R., Di Ruscio, D., Nguyen, P.T.: in *Proceedings of the Evaluation and Assessment in Software Engineering* (Association for Computing Machinery, New York, NY, USA, 2020), EASE '20, p. 71-80. https://doi.org/10.1145/3383219.3383227. https://doi.org/10.1145/3383219.3383227
6. Zhou, Y., Wu, J., Sun, Y.: GHTRec : A Personalized Service to Recommend GitHub Trending Repositories for Developers, 2021 IEEE International Conference on Web Services (ICWS) pp. 314–323 (2021). https://doi.org/10.1109/ICWS53863.2021.00049
7. Di Rocco, J., Di Ruscio, D., Di Sipio, C., Nguyen, P.T., Rubei, R.: Hybridrec: A recommender system for tagging github repositories. Appl Intell **53**(8), 9708–9730 (2023). https://doi.org/10.1007/s10489-022-03864-y
8. Zhang, Y., Lo, D., Kochhar, P.S., Xia, X., Li, Q., Sun, J.: Detecting similar repositories on GitHub, SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering pp. 13–23 (2017). https://doi.org/10.1109/SANER.2017.7884605
9. Bai, S., Liu, L., Liu, H., Zhang, M., Meng, C., Zhang, P.: Find potential partners: A GitHub user recommendation method based on event data. Information and Software Technology **150**(May), 106961 (2022). https://doi.org/10.1016/j.infsof.2022.106961
10. Borges, H., Hora, A., Valente, M.T.: Understanding the factors that impact the popularity of GitHub repositories, Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016 pp. 334–344 (2017). https://doi.org/10.1109/ICSME.2016.31
11. Zhou, S., Vasilescu, B., Kästner, C.: How has forking changed in the last 20 years? a study of hard forks on github, Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering p. 445-456 (2020). https://doi.org/10.1145/3377811.3380412
12. Venigalla, A.S.M., Chimalakonda, S.: An exploratory study of software artifacts on github from the lens of documentation. Information and Software Technology **169**, 107425 (2024). https://doi.org/10.1016/j.infsof.2024.107425. (https://www.sciencedirect.com/science/article/pii/S0950584924000302)
13. Zanartu, F., Treude, C., Cartaxo, B., Borges, H.S., Moura, P., Wagner, M., Pinto, G.: Automatically categorising github repositories by application domain, arXiv preprint arXiv:2208.00269 (2022). https://arxiv.org/abs/2208.00269. Accessed: 2025-04-05
14. Mahn, A., Topper, D., Quinn, S., Marron, J.: Getting started with the nist cybersecurity framework: A quick start guide (2021). https://doi.org/10.6028/NIST.SP.1271. (https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=932713)
15. Wang, S., Manning, C.D.: Baselines and bigrams: Simple, good sentiment and topic classification, The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, Volume 2: Short Papers pp. 90–94 (2012). https://aclanthology.org/P12-2018/
16. McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction, arXiv preprint arXiv:1802.03426 (2020). https://arxiv.org/abs/1802.03426. Accessed: 2025-04-05
17. Campello, R.J.G.B., Moulavi, D., Zimek, A., Sander, J.: Hierarchical density estimates for data clustering, visualization, and outlier detection, ACM Trans. Knowl. Discov. Data **10**(1) (2015). https://doi.org/10.1145/2733381
18. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm, Journal of the Royal Statistical Society. Series C (Applied Statistics) **28**(1), 100 (1979). http://www.jstor.org/stable/2346830
19. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval, vol. 39. Cambridge University Press Cambridge, Cambridge, England (2008)
20. Ye, J., Chen, X., Xu, N., Zu, C., Shao, Z., Liu, S., Cui, Y., Zhou, Z., Gong, C., Shen, Y., Zhou, J., Chen, S., Gui, T., Zhang, Q., Huang, X.: A comprehensive capability analysis of gpt-3 and gpt-3.5 series models, arXiv preprint arXiv:2303.10420 (2023). https://arxiv.org/abs/2303.10420. Accessed: 2025-04-05