2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) | 979-8-3503-0037-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-SEIP58684.2023.00050

An Empirical Study of License Conflict in Free and Open Source Software

Xing Cui*, Jingzheng Wu*,#, Yanjun Wu*, Xu Wang*, Tianyue Luo*, Sheng Qu*,
Xiang Ling*,# and Mutian Yang†
*Institute of Software, Chinese Academy of Sciences, †Beijing ZhongKeWeiLan Technology Co.,Ltd.
{cuixing, jingzheng08, yanjun, wangxu, tianyue, qusheng, lingxiang, mutian}@iscas.ac.cn

***Abstract*—Free and Open Source Software (FOSS) has become the fundamental infrastructure of mainstream software projects. FOSS is subject to various legal terms and restrictions, depending on the type of open source license in force. Hence it is important to remain compliant with the FOSS license terms. Identifying the licenses that provide FOSS and understanding the terms of those licenses is not easy, especially when dealing with a large amount of reuse that is common in modern software development. Since reused software is often large, automated license analysis is needed to address these issues and support users in license compliant reuse of FOSS. However, existing license assessment tools can only identify the name and quantity of licenses embedded in software and thus cannot identify whether the licenses are being used safely and correctly. Moreover, they cannot provide a comprehensive analysis of the compatibility and potential risk that come with the term conflicts.**

In this paper, we propose DIKE, an automated tool that can perform license detection and conflict analysis for FOSS. First, DIKE extracts 12 terms under 3,256 unique open source licenses by manual analysis and Natural Language Processing (NLP) and constructs a license knowledge base containing the responsibilities of the terms. Second, DIKE scans all licenses from the code snippet for the input software and outputs the scan results in a tree structure. Third, the scan results match the license knowledge base to detect license conflicts from terms and conditions. DIKE designs two solutions for software with license conflicts: license replacement and code replacement. To demonstrate the effectiveness of DIKE, we first evaluate with the term extraction and responsibility classification, and the results show that their F1-scores reach 0.816 and 0.948, respectively. In addition, we conduct a measurement study of 16,341 popular projects from GitHub based on our proposed DIKE to explore the conflict of license usage in FOSS. The results show that 1,787 open source licenses are used in the project, and 27.2% of licenses conflict. Our new findings suggest that conflicts are prevalent in FOSS, warning the open source community about intellectual property risks.

***Index Terms*—Free and Open Source Software, license analysis, license conflict, natural language processing**

I. Introduction

Free and open source software (FOSS) plays a significant and necessary role in the software development process for developers to build and deliver new software products at an unprecedented rate. Currently, FOSS has been widely adopted by academia and industry and extensively deployed by commercial products in its production cycle due to recent technical breakthroughs, which can significantly raise the development efficiency and reduce the production cost [1]–[3]. Although the open source code is usually free to use, open source licenses that it comes with usually specify a set of terms or restrictions that developers must confirm [4]–[6]. Consequently, the redistribution and commercialization of new software will be negatively impacted if the contradicted terms or violations are found within the FOSS that has been used by the developers. For example, if the open source code is mixed with the non-open source code under copyright by a third party, the developer mistakenly believes that the whole software is open source code without paying the license fee, then there is a risk of copyright infringement. Nonetheless, the lack of a reliable and efficient licensing violation assessment system resulted in the issue remains unsolved.

While the open source community encourages developers and organizations to use open source components, combining a number of licenses can be a challenge since different license terms can conflict. Sometimes a developer will combine multiple components under different open source licenses with direct or transitive dependents, raising the legal concern of whether their licenses are compatible. For instance, the *ChatSecureAndroid project* declares that Apache-2.0 is used, but its subfile *CertificateChainCleaner.java* declares to use GPL-3.0. Since the above two licenses do not share the "*Disclose source*" term, they may cause a violation when developers decide to make the project open source. The GNU website currently details what open source licenses are compatible with the GNU General Public License (GPL). However, these licenses do not fully cover all of the current major licenses [7]. While expressed in natural language, the licenses do not follow convention. Since licenses are not easily interpreted, it is necessary to rely on manual text analysis if compatibility between licenses is to be determined [8]. As observed earlier, developers do not fully understand the consequences of adding specific licenses to their code (or not adding any licenses) and may encounter intellectual property violations.

Although there are license assessment tools that can be leveraged on the market, those tools all have distinct limitations such as limited discriminable characteristics and license volumes [9], [10]. For instance, the open source license compliance package FOSSology performs pattern matching on the uploaded source code to find a matching license. Ninka [11] can identify and classify licenses in software projects. The Licensefinder analyzes the license of the code

#Jingzheng Wu and Xiang Ling are the corresponding authors.

2832-7659/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-SEIP58684.2023.00050

in the project, but requires a user-defined safe list of available licenses to perform matching, and then reports the operation. Some tools include license compatibility detection functions, such as FindossLicense [12], which can select the appropriate open source license for the software being created to some extent. In summary, previous works focus on identifying license names and listing the information identified on files or code fragments, rather than analyzing compatibility between the licenses. The existing assessment tools have limitations in large-scale software development [13]. As a result, it has become an urgent problem to design an automatic license compatibility analysis method.

In this work, we present DIKE, an automated tool for licensing conflict analysis, in order to investigate the current state of license usage in FOSS, to solve the prevalent license conflict issue, and to aid in the research of the intellectual property rights of the open source community. DIKE is capable of scanning open source licenses for software in all programming languages. By matching the scan results with the license knowledge base, it detects conflicts in using open source licenses. In addition, DIKE provides two ways to resolve conflicts: code replacement and license replacement. In DIKE, we build a term extraction model based on a Label-Specific Attention Network (LSAN) [14], inspired by multi-label text classification, and use the semantic information of labels to determine the semantic linkage between terms and licenses. Notably, we borrowed focal loss to improve the loss function [15], which alleviates category imbalance and sample imbalance in multi-class classification. To make judgments on the responsibility of terms, we use ALBERT [16] to classify. To obtain the distribution of each license in the software intuitively, we output the scanning results of licenses in a tree structure with semantic information to locate the locations where the licenses appear. DIKE matches the detection results with the license knowledge base, makes a conflict determination based on the calculation results, and designs code replacement and license replacement to resolve license conflicts. We select some active conflict projects and give feedback to the developers about the conflicts. Some of them accept our suggestions (The *rbenv* project is a typical example, available at https://github.com/rbenv/rbenv/issues/1395). In summary, this paper makes the following contributions:

- Using NLP, we offer the licensing conflict analysis tool DIKE, which can automatically extract the 12 terms of open source licenses and establish the responsibility of the terms, considerably decreasing the manual workload of license analysis.
- A "License-Terms-Responsibility" knowledge base is created by DIKE's analysis of 3,256 unique open source licenses; this database is then used to help scan software licenses, identify usage conflicts, and provide feasible solutions to these problems. Based on our research, this conflict analysis tool supports the largest range of open source licenses and places no limitations on the programming language used by the detected project.
- We analyze 16,341 open source projects with more than 1,000 stars on GitHub for license conflicts. All projects use 1,787 open source licenses, with 4,448 (27.2%) conflicting projects. The findings serve as a warning for open source compliance studies, revealing that a sizable proportion of projects suffer from the issue of improper license usage.

II. Background

A. Open source License

An open source license is a type of license for computer software and other products that allows the source code, blueprint, or design to be used, modified and shared under defined terms and conditions [17], [18]. Open source licenses dictate the terms and conditions that come with the use of FOSS. Open source licenses serve as a legal agreement between open source author and user: authors make software available for free, but with specific requirements the user must follow [19]. In today's mainstream open source licenses on the market, the stated terms relate to the software rights and interests mainly include *Commercial use, Distribution, Modification, Patent use, Private use*, and *Same license*, etc. Although the licenses in each category share common characteristics, the license terms give more details on what is allowed or not. In reference to the license description provid by *choosealicense* [20], the following information refers to the terms and condition.

- Commercial use: The licensed material and derivatives may be used for commercial purposes.
- Distribution: The licensed material may be distributed.
- Modification: The licensed material may be modified.
- Patent use: (1) This license provides an express grant of patent rights from contributors. (2) This license explicitly states that it does NOT grant any rights in the patents of contributors.
- Private use: The licensed material may be used and modified in private.
- Same license: Modifications must be released under the same license when distributing the licensed material. In some cases a similar or related license may be used.

B. Conflict of open source License

Open source licenses vary according to their obligations on users and the permissions they grant. Based on these parameters, open source licenses are divided into two main types: permissive and copyleft. Copyleft licenses allow users to use, modify, and share works to maintain reciprocal obligations. Components with copyleft licenses also require users to make their code available to others [21]. On the other hand, the permissive license gives users the freedom to use, modify, and redistribute and allows proprietary derivative works with minimal obligations or restrictions imposed on the user [22]. Even though copyleft and permissive allow developers to use the software without paying the owner, there are specific differences in licensing authority. The main difference is the difference in the disposition of the code

by the person using the open source code. GPL (2.0, 3.0), LGPL, AGPL, and other agreements are copyleft types of open source protocols [23]. On the other side, permissive licenses have much fewer restrictions. The developers can modify and use the code they desire, including *Commercial use*. Copyleft licenses prohibit proprietary software from using their code and explicitly require that all programs use the same licenses. In a combined program, the permissive and copyleft licenses are usually integrated, and the copyleft license is required for the combined program as a whole. However, some licenses do not fall into the two categories mentioned above. In this case, the compatibility must be determined based on the declared terms and conditions. As long as the terms and conditions in the licenses are compatible, combining two or more different open source components will not have conflict. Although the open source community encourages developers and organizations to reuse open source components, under the premises of avoiding legal risks while combining multiple licenses can still be a challenge.

C. Conflict problem solution

In recent years, some organizations have developed metrics to manage and measure the conflict in FOSS development. Among them, few studies on the compliance with intellectual property rights related to FOSS. As described by Zhang et al. [24], their compliance checking method provides an automated solution to help developers detect license conflicts. However, the scope of this tool on the market has lots of limitations. Possible casual discussions about intellectual property and FOSS were elaborated in the Open Business Readiness Evaluation Forum. One of the research and development areas of the Commercial Open Source Software Quality Platform project is to provide guidelines and tools to facilitate the intellectual property tracking process of FOSS and define a consistent license that complies with national laws and European regulations [25]. In addition, there are some representative tools. Black Duck software composition analysis allows users to discover and identify the open source components in proprietary software, their corresponding open source licenses, and vulnerabilities to help mitigate legal and security risks [26]. The FOSSology recommends analyzing all source code of a given project and intelligently reporting all licenses being used based on the license statement and descriptive phrases to help users identify the software license [27].

III. Methodology

A. Overview

As illustrated in Figure 1, the proposed DIKE consists of three key components: (1) License Analyzer; (2) License Sniffer; (3) Solution Engine. The License Analyzer processes the license text to build a knowledge base covering "License-Terms-Responsibility" by extracting terms, classifying terms, determining usage propensity, and automatically processing any new license. The License Sniffer attempts to detect licenses of software. The Solution Engine then traverses the relational knowledge base of all the licenses detected in the software to determine license conflicts and provide further solutions for software with conflicting licenses.

B. License Analyzer

For the wide variety of existing licenses, there are limitations in manual analysis [28], [29]. The License Analyzer component uses NLP techniques and specific text representations to automate the analysis of license agreement terms. The component consists of two steps: license terms extraction and terms responsibility recognition. We define 12 categories corresponding to each of the 12 terms commonly found in licenses, including *Same license, Patent use, Disclose source, Commercial use, Trademark use, Distribution, Private use, Modification, License and copyright notice, State change, Liability, Warranty*. The outputs of License Analyzer are the input terms and responsibility for this license term, with the possible values of permitted (Y) or warranty (N).

Multiple terms usually appear in one short paragraph in the license text, multi-label classification can be used to classify a single paragraph into multiple categories. The text content and tag content are fully exploited to learn the text representation, facilitating sharing of the same subset of texts [9]. In this paper, we combine the characteristics of license text and construct a multi-classification model for recognizing terms based on LSAN. The model structure is shown in Figure 2.

The multi-label classification has two parts. The first part adaptively captures information related to the text content and the label from each content and label content; the second part extracts information from these two aspects. Finally, the classification model is trained on the text representation incorporating specific tags.

Due to a large number of multi-label situations in the license text, we use the self-attention mechanism of semantic mining to assign different labels to the text in the multi-classification stage.

$$A^{(s)} = softmax(W_2 \tanh W_1 H) \tag{1}$$

In which W_1 and W_2 are training parameters for self-attention. Each row $A_j^{(s)}$ represents the contribution of all words to the j label. Then we can obtain the label-word score to obtain the linear combination of each label context:

$$M_j^{(s)} = A_j^{(s)} H^T \tag{2}$$

Based on the self-attention mechanism stage, the representation can be obtained through the text. Because tags have specific semantics, they are usually implicit descriptions of text content. Calculate the semantic relationship between words and tags, and construct tags to represent specific document content by linearly combining the upper and lower words of the tags. This label-text-based attention mechanism is expressed as:

$$M^{(l)} = (\overrightarrow{M}^{(l)}, \overleftarrow{M}^{(l)}) \tag{3}$$

The text-based attention mechanism and the semantically associated tag-text-based attention mechanism respectively focus on the text content and the semantic association between

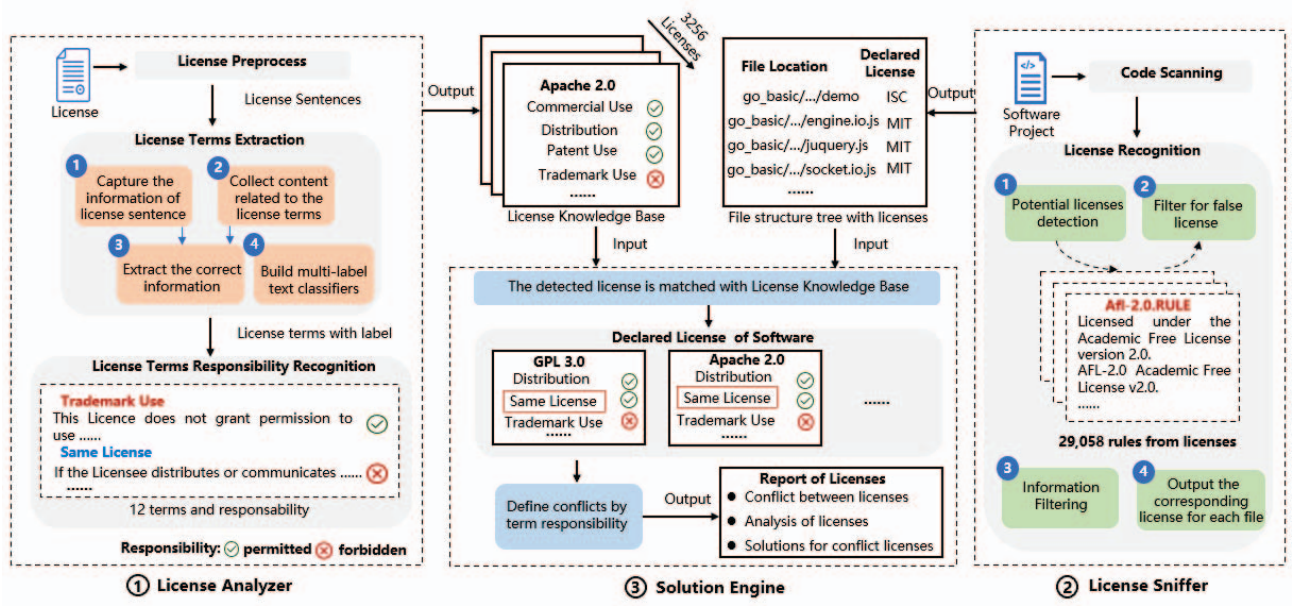

Fig. 1. The overview of DIKE. DIKE consists of three components: (1) License Analyzer processes the license text to build a knowledge base, covering "License-Terms-Responsibility". (2) License Sniffer can detect licenses of software. (3) Solution Engine traverses and matches the knowledge base in (1) with the detection results in (2) to determine license conflicts and provide solutions for software with license conflicts.

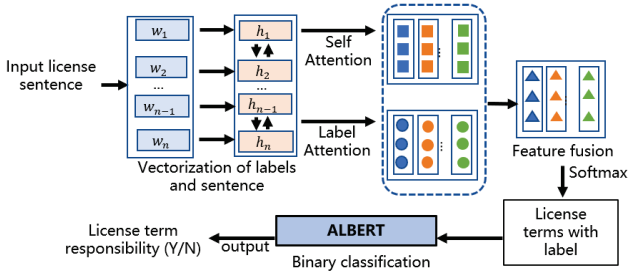

Fig. 2. Workflow in License Analyzer

texts. We use feature fusion to extract corresponding information features to construct a comprehensive representation of specific text content. We put the weights of text and labels into the fully connected layer. Through the fusion strategy, we obtain the text content representation along the j label as:

$$M_j = sigmoid(M_j^{(s)} W_3) M_j^{(s)} + sigmoid(M_l^{(s)} W_4) M_j^{(l)} \tag{4}$$

The text representation uses the $relu$ transformation and the focal loss as the objective function for multiple classifications. Using the focal loss to solve the problem of severe imbalance in the ratio of positive and negative samples in training samples to a certain extent [30]. This loss function reduces the weight of a large number of simple negative samples in training.

After the license text is classified with multiple labels, it is also necessary to make a judgment corresponding to each label. We use ALBERT for binary classification. ALBERT applies the attention mechanism to extract information about the context of a given word and then encodes it into the learning vector. A sentence is divided into N characters, and N embeddings are used to correspond, which are $E_1, E_2, ..., E_n$, using tokenization involves segmenting the input text into a list of available signatures in the vocabulary. Using the signature $[CLS]$ to mark the beginning of a sentence and $[SEP]$ to separate different sentences. Take the value corresponding to $[CLS]$ and go through a fully connected layer to get the classification result. The binary classification outputs the terms and possible value for responsibility (Y/N). The License Analyzer eventually outputs the license terms and associated responsibilities by combining the results of the two classifications. Based on the output of License Analyzer, we build a license knowledge base containing "License-Terms-Responsibility".

C. License Sniffer

The workflow of License Sniffer mainly consists of four parts, File Scanning, Potential license detection, Precise matching, and Information filtering.

File scanning. When a code file is to be scanned, the first step is to convert it into a query object. We propose an exploratory query approach that divides the query object into multiple word slices. Then each word will be mapped to an integer. Therefore, the text containing licenses is transformed into a sequence of numbers with potential patterns. The text containing licenses is abstracted into a lexicon where the corresponding sequence of numbers is also specific. To ensure the usability of the text, a threshold to the text segment was assigned. When the value of a text segment is lower than the

predefined threshold, the text segment cannot be considered as a reference to a license.

Potential license detection. The matching process is queried against the hash value of the whole text, and then the mapping to the license rule is found again according to the hash table [31]. We have collected 29,058 license rules for mapping. For example, for Apache-2.0,

"Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at https://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License." If the scanned text matches one of the rules, which means a license that converts all processes into an operation between integer vectors may exist. Any exact matches to "negative rules" from the query that do not contain license information but may match existing rules are removed. We then use an Aho-Corasick algorithm to perform exact matches on the entire query [32]. Aho-Corasick is the Multipattern matching algorithm which locates all the occurrences of a set of patterns in a text of string. It first creates deterministic finite automata for all the predefined patterns, and then by using automaton, it processes a text in a single pass. It consists of constructing a finite state pattern matching automata from the patterns and then using the pattern matching automata to process the text string in a single pass [33].

Precise matching. In order to detect the licenses more accurately, we add two more rounds of precise matching. The first round is set matching. It adopts the bag-of-words approach. The text is first scanned and converted into a vector of integers based on the presence or absence of keywords. The concept applied here is similar to the inverted index in information retrieval to the second round comparison among integer vectors. The second round of comparison is performed among the best matching sets. Finding the best matching set in the first round, using similarity and inclusion to sort together after getting the number of times each word appears in the text.

Information filtering. After the above two rounds of matching are completed, several candidate matches are selected, and all the collected matches are merged and filtered to produce the final result. The merging takes into account the similarity, inclusion, and overlap between the scanned text and the matched text. Filtering is based on the density and length of the matches and the number of good or frequent tokens matched. Finally, the refined, filtered matches are derived based on the similarity, inclusion, and overlap between the query text and the matched text. Finally, the matching results are filtered to output information about the licenses, including name, number, type, etc. Through the correspondence between the matched files and licenses and the directory information where the project files are located, a more detailed distribution of license usage and the corresponding hierarchical dependencies of the whole project is obtained. As shown in Figure 3, License Sniffer outputs a file structure tree with licenses.

```
[License name: mulanpsl-2.0][go_basic]
|——[License name: isc][go_basic/socket.io/demo]
|    |——[License name: mit][go_basic/socket.io/demo/web/asset/engine.io.js]
|    |——[License name: mit][go_basic/socket.io/demo/web/asset/jquery.js]
|——[License name: mit][go_basic/socket.io/static/js/socket.io.js]
```

Fig. 3. File Structure Tree

D. Solution Engine

Solution Engine traverses the licenses scanned by the License Sniffer component in the license knowledge base and determines them based on the "License-Terms-Responsibility" relationship. Once the same term under two licenses has a different responsibility, we consider a potential conflict between the two licenses. If one of the same terms in the different licenses is Y and the other is N, then this situation can only be considered as a possible conflict, so further judgment is needed based on the location of the license file in the license structure tree. The last term in the pairwise comparison should be a strict one under two usage conditions. If this term exists in the license of the project statement, no conflict is considered to exist. For example, the project declaration uses GPL-3.0, and Apache is quoted in the code. Although the declarations on the responsibility of *Patent use* are different, the GPL-3.0 requirement is more valid, and the two are not conflicted in this case.

For projects with conflicting licenses, we design two solutions, code replacement and license replacement. For license replacement, first, evaluate the combination of conflicting licenses and create a list of all strictly required terms contained in the licenses. Then, in the license knowledge base, search for a license that includes the set of terms and has a strict responsibility that can be used to substitute the conflicting license in the combination. If code replacement is used, it is necessary to determine whether the code containing the conflicting license is critical. If the code is vital, the developers should restructure it according to its function or replace it with code that performs the same function but does not have a licensing conflict. If the code is non-critical and does not influence the software's functionality, it can be considered for deletion.

IV. Evaluation Settings

A. Evaluating the License Analyzer

License Analyzer is the core of DIKE, and its performance affects the accuracy of conflict identification. We collect 3,256 unique licenses and manually label 319 of them at random. Manual labeling consists of two stages: (1) deleting blank lines, structuring each license text into paragraphs, and then slicing each paragraph into individual sentences. (2) The labeler denotes the sentence's term category and responsibility

based on its semantics and descriptions. Eight of our authors accomplish the manual annotation, which two lawyers then check. And after that, We randomly split these samples into the training and testing datasets by 4:1 and utilize them for training and testing. In addition to the manually annotated licenses, we input the remaining 2,937 licenses collected into License Analyzer for automatic processing and output the terms and corresponding responsibilities. Finally, we build the output into a license knowledge base, which contains the automated and manual analysis results of 3,256 existing open source licenses.

We perform two classifications to build the license analysis model, namely license terms extraction and terms responsibility recognition. We use three evaluation metrics on the license terms extraction: (1) Precision represents the fraction of the relevant terms over the extracted terms; (2) Recall represents the fraction of the relevant terms that are extracted over the total number of relevant terms; (3) F1-score represents the harmonic mean of precision and recall. In addition to the above three metrics, the terms responsibility recognition adds overall accuracy for evaluation. Overall accuracy represents the fraction of the correct predictions over all the predictions. The evaluation is based on true positive (t_p) true negative (t_n) false positive (f_p) and false negatives (f_n). Among them, $Precision = tp/(tp + fp)$, $Recall = tp/(tp + fn)$, $F1 = (Precisoin * Recall * 2)/(Precision + Recall)$. A multi-label classification can be regarded as multiple binary classifications, and these indicators can be calculated on the entire test data through macro averaging. Macro average refers to the arithmetic average (Precision, Recall, and F1-score) of each statistical indicator value of all categories.

$$Precision_{macro} = \frac{1}{n}\sum_{i=1}^{n} Precision_i \tag{5}$$

$$Recall_{macro} = \frac{1}{n}\sum_{i=1}^{n} Recall_i \tag{6}$$

$$F1_{macro} = \frac{2 \times Precision_{macro} \times Recall_{macro}}{Precision_{macro} + Recall_{macro}} \tag{7}$$

License terms extraction is a multi-label classification model. Our multi-label classification model is based on LSAN and introduces focal loss for data imbalance. In the training process, the default batch size is 16, and the number of epochs is 300 with an early stop. We deploy Adam's learning rate to be 0.0001 and the weight decay to 5e-5. Considering the problem of data imbalance, the focal loss is used as the loss function, $focal\_loss\_alpha = 0.5$, $focal\_loss\_gamma = 2$. We perform model training on TITAN RTX GPU with 24 GB of memory. Table I shows our model and baselines' macro Precision, macro Recall, and macro F1-score. License terms can be extracted with a macro Precision of 0.886, a macro Recall of 0.766, and a macro F1-score of 0.816. The term responsibility recognition is a binary classification model based on ALBERT. Table II shows the result of the terms responsibility, with an overall Accuracy of 0.943, Precision of 0.937, Recall of 0.960, and F1-score of 0.948.

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT MODELS IN LICENSE TERMS EXTRACTION

Model	macro Precision	macro Recall	macro F1-Score
ALBERT	0.839 (±0.021)	0.765 (±0.035)	0.782 (±0.027)
ALBERT+Focal Loss	0.883 (±0.026)	0.755 (±0.014)	0.795 (±0.014)
LSAN	0.806 (±0.019)	0.756 (±0.023)	0.767 (±0.016)
LSAN+Focal Loss	0.886 (±0.018)	0.766 (±0.022)	0.816 (±0.014)

TABLE II
PERFORMANCE OF MODELS IN LICENSE TERMS RESPONSIBILITY RECOGNITION

Model	Accuracy	Precision	Recall	F1-Score
ALBERT	0.943 (±0.008)	0.937 (±0.014)	0.960 (±0.011)	0.948 (±0.007)

We validate the effectiveness of the License Analyzer and the correctness of the license knowledge base through three sets of randomized controlled trials. We randomly select 50 licenses analyzed by License Analyzer in the knowledge base and have three authors manually annotate the terms and the corresponding responsibilities without knowing the analysis results and cross-validate them. Then, the manual analysis results are compared with the processing results of License Analyzer. To avoid the probability of the experimental results, we repeat the above experiment three times. The average number of terms analyzed manually in the 3 trials is 262, as shown in Table III, and the average number of terms analyzed by License Analyzer is 259. The average number of terms accurately analyzed by license analyzer is 201, with a 76.7% accuracy rate and a 0.772 F1-score.

TABLE III
PERFORMANCE COMPARISON OF THE LICENSE ANALYZER WITH MANUAL ANALYSIS

ID	Manual	DIKE	Accuracy	Precision	Recall	F1-Score
1	258	243	194 (75.2%)	0.798	0.752	0.774
2	263	257	196 (74.5%)	0.763	0.745	0.754
3	264	276	213 (80.7%)	0.772	0.807	0.789
AVG	262	259	201 (76.7%)	0.778	0.768	0.772

B. Baseline comparisons

To compare the performance of License Analyzer against other baselines, we apply other methods to the same dataset.

First, for the license terms extraction model, we test ALBERT. ALBERT uses the Sentence Order Prediction (SOP) task instead of the Next Sentence Prediction (NSP) task in BERT to achieve state-of-the-art results in several natural language understanding tasks. We find that ALBERT achieves a reasonable macro F1-score of 0.782. Based on the use of ALBERT, we choose Focal Loss [15] to modify the loss function of ALBERT to prevent the overlearning of samples which easy to classify. The new loss function makes the model focus more on the samples during training by reducing the

weights of easy-to-classify samples. The evaluation returns a macro F1-score of 0.795.

Second, we implement a baseline system using LSAN. The LSAN uses label-related attention networks to learn document representations. Semantic information of the labels is then used for constructing the document representation to determine the semantic association between the labels and the documents. In addition, self-attention is used to identify label-specific document representations based on document content information. To integrate the above two parts, an adaptive fusion is used to output a comprehensive document representation. The LSAN with the macro F1-score of 0.767. Our multi-label classification model achieves a macro F1-score of 0.816, which shows a 3.4%, 2.1%, and 5.5% improvement compared to ALBERT, ALBERT+Focal Loss, and LSAN, respectively.

V. Evaluation Results and Analysis

To explore the usage of licenses in open source projects, conflict issues, and the severity of conflicts, we conduct a large-scale measurement study of over 1,000 star projects on GitHub. We use DIKE to detect the licenses used by these projects, analyze the conflicts that exist in the license combinations, and give valuable recommendations. And the following research questions are answered.

RQ1: Which licenses are used in open source projects? We detect the use of licenses in DIKE's collected open source projects.

RQ2: Are there any license conflicts in open source projects? We match the detected project licenses with the license knowledge base and find which projects have conflicting license combinations.

RQ3: How to resolve license conflicts? Combined with using licenses in open source projects, our goal is to provide a reasonable solution.

To answer RQ1, we use DIKE to find randomly selected GitHub projects with over 1000 stars and analyze them:

- The open source license the project declares to use.
- The open source license the files within the project declare to use.
- The open source license the project and its files prefer to choose.

To answer RQ2, we compare the license portfolios of all of the projects in RQ1 to the license knowledge base maintained by DIKE. We calculate:

- The percentage of projects with conflicting licenses.
- The licenses are most closely related to projects with conflicts.
- The license terms are most closely related to conflicts.
- The potential causes of license conflicts.

To address RQ3, we explore two solutions while performing ongoing validation in the projects collected in RQ1 and committing the findings and solutions on GitHub.

RQ1: Which licenses are used in open source projects?

We collect 16,341 FOSS projects with more than 1,000 stars on GitHub. The programming languages for these projects include C, Java, Python, C++, Python, PHP, SQL, JavaScript, and 203 others. There are 14,624 projects that have created the LICENSE files (LICENSE.txt, LICENSE.md or LICENSE.rst). In addition, a few projects (10.5%) do not declare a license, and no one is permitted to reproduce, distribute, or make derivative works from these. Using DIKE, we analyze the collected projects and discover 139 different licenses via LICENSE files. In addition, we extract 9,571,954 files from the gathered projects and analyze the license information contained in the code comment language of the retrieved files; a total of 1,387 different licenses are identified from a vast number of recovered files (including LICENSE files) .

Many of the licenses identified in the dataset belong to the same series. For instance, there are 172 GPL licenses, 27 Apache licenses, and 103 BSD licenses. Table IV displays the ten most popular licenses ordered by the number of projects using each license. The majority of projects (79.8%) choose popular licenses, with MIT, Apache-2.0, and GPL-3.0 being the most popular. According to the statistics, the MIT license is more prevalent when declaring the project's LICENSE, which currently covers 62.6% of projects. The MIT license is concise, straightforward, and contains the fewest limitations. The Apache license is comparable to the MIT license but includes provisions for contributors to provide patent licenses to consumers. The GPL license is more restrictive than the licenses mentioned above, as it requires a user who modifies the source code of a project to publish his modifications before redistributing the source or binary code.

TABLE IV
Ten most popular licenses by the amount of project

License	Project Number	Type
MIT	7,375	permissive
Apache-2.0	2,859	permissive
GPL-3.0	883	copyleft
BSD-3-Clause	829	permissive
BSD-2-Clause	248	permissive
GPL-2.0	240	copyleft
AGPL-3.0	225	copyleft
GPL-3.0-or-later	128	copyleft
MPL-2.0	126	copyleft
ISC	124	permissive

According to the number of files, Table V lists the ten most popular licenses, with Apache-2.0, MIT, and BSD-3-Clause being used more frequently in the files. All three of these licenses permit the creation of derivative works, commercial use, and distribution. Apache-2.0 is the license used most frequently in the documentation. In contrast to MIT and BSD-3-Clause, Apache-2.0 offers the user a copyright license in addition to a patent license, allowing the user to receive a perpetual license but requiring the placement of a copyright notice whenever modifications are made. Apache-2.0 is better suited for business applications without open source and with relatively strict conditions. Legal risk is relatively low for commercial enterprises, likely one reason for its growing popularity.

TABLE V
TEN MOST POPULAR LICENSES BY THE NUMBER OF FILES

License	File Number	Type
Apache-2.0	1,121,565	permissive
MIT	731,832	permissive
BSD-3-Clause	307,726	permissive
GPL-2.0-or-later	116,561	copyleft
GPL-3.0-or-later	107,090	copyleft
EPL-1.0	73,813	copyleft
GPL-2.0	65,201	copyleft
BSD-2-Clause	47,212	permissive
Boost-1.0	41,072	permissive
GPL-3.0	38,782	copyleft

TABLE VI
STATISTICS OF THE LICENSE CONFLICT

License Conflict Pair	Project Number	Propertion
MIT with GPL-3.0-or-later	393	2.40%
MIT with GPL-2.0	372	2.27%
Apache-2.0 with GPL-3.0-or-later	232	1.41%
Apache-2.0 with GPL-2.0	227	1.38%
MIT with GPL-3.0	199	1.20%
MIT with GPL-2.0-or-later	169	1.03%
GPL-3.0 with GPL-2.0	146	0.89%
BSD-3-Clause with GPL-3.0-or-later	138	0.84%
Apache-2.0 with MPL-2.0	125	0.76%
GPL-2.0 with GPL-3.0-or-later	125	0.76%

TABLE VII
STATISTICS ON THE USE OF OPEN SOURCE LICENSE TERMS

License terms	Conflict Project	Propertion
Same license	4,148	25.38%
Patent use	2,643	16.17%
Disclose source	2,087	12.77%
Commercial use	407	2.49%
License and copyright notice	208	1.27%
Private use	150	0.92%
Trademark use	82	0.50%
Distribution	27	0.17%
Modification	4	0.02%
State changes	/	/
Liability	/	/
Warranty	/	/

RQ2: Are there any license conflicts in open source projects?

We utilize DIKE to detect licensing conflicts based on the data collected by RQ1. The results indicate that 4,448 projects contain conflicts, which is greater than a quarter of the total number observed (27.2%). We also note that only 95 items conflict without LICENSE files. The ten licensing pairings are arranged by the number of conflicting entries in Table VI. The data indicate that approximately half of the disputes are produced by combinations of the GPL family and open source licenses, with the most significant conflicts arising between the MIT, Apache-2.0, and GPL families. This contradiction is caused by the "infectious" nature of the GPL. For instance, GPL-2.0 permits later developers to edit the FOSS (and their copies). Still, the modifications or derivative works that come from such modifications can be used to produce a new version of the program after they are published or distributed. For instance, GPL-2.0 permits subsequent developers to alter the software (and their copies), but once modified versions or derivative works are distributed or published, they must be made open source by the applicable GPL agreement. In addition, under certain circumstances, FOSS might "infect" itself and other software sections or software that are distributed or transmitted with the modified version. For instance, GPL-3.0 stipulates that if the integration of FOSS with other software is intended to result in a more extensive program when the developer transmits the software, then the other software must be infected with the matching GPL protocol. The GPL's infection term mandates that any distributed or released works (including all or a portion of this program or any part thereof, modified or not) must be distributed under the conditions of this general rule. A violation of the contagion term renders the copyright license (free copyleft) granted through the open source agreement null and void.

The terms used in the licenses of the conflicting projects are shown in Table VII. Most conflicting projects are associated with "*Same license, Patent use, and Disclose source*". The *Same license* is a restrictive term that requires the software to be distributed with the same license to distribute modifications. However, similar or related licenses may be used in certain circumstances. In other words, if program *A* using license I must extend license I and program *B* using license II must extend license II, then they are irreconcilable; once the combined program contains the code of *A* and *B*, then the license of the combined program must be I and II, but this does not meet the requirements of the *Same license*, causing a license conflict. Because they do not meet the *Same license* term, GPL-2.0 and GPL-3.0 are incompatible. Another sort of term that arises in a large number of conflicting projects is *Patent use*. One of the main reasons for this is that Apache-2.0 and GPL-2.0 are used in many combinations in projects, and Apache-2.0 has a patent term incompatible with GPL-2.0. The source code must be made accessible when software is released to enable open source. This is referred to as the *Disclose source* term.

RQ3: How to resolve license conflicts?

In RQ2, we discover that a high percentage of open source projects have license conflicts. DIKE offers two solutions to resolve these license conflicts: (1) Replace the open source license; (2) Replace the source code. For solution (1) , When DIKE identifies project conflicts, it compiles a list of all strict responsibility terms of these licenses and searches the license knowledge base for licenses that contain these terms and all have strict responsibility, which can be used to substitute conflicting licenses in the project. Solution (2) is to replace the code in the project with conflicting licenses with code of the same purpose but with non-conflicting licenses, or the project's creator can rewrite the code to meet the functional requirements.

We continue to provide feedback on conflict issues found in RQ2 in related GitHub projects, with a total of 207 issues submitted so far, of which 59 active projects have accepted our suggestions and further fixed conflict issues. The project with the highest number of stars that have accepted our conflicting issues and suggestions for improvement is *rbenv*. *rbenv* is a popular version manager tool for the Ruby programming language with a current star count of 14.4k. It is useful for switching between multiple Ruby versions on the same machine and ensuring that each project working on always runs on the correct Ruby version. We have detected a license conflict for this project. The file *shobj-conf* in this project is released under the GPL-3.0-or-later license. According to the GPL-3.0 license, the overall project also needs to be released under GPL-3.0-or-later license. The problem is that this project is licensed under the MIT license. For this project, MIT and GPL-3.0-or-later are used, leading to a license conflict. Figure 4 shows the license conflict for *rbenv* and the solution we provide. In the end, the developer chose to remove the conflicting files from the project, resolving the conflict.

Figure 5 shows the response of some projects on conflict issues. While utilizing the responses, we learn that some developers aren't apprehensive about the conflict concerns or believe these conflicts have less of an impact on the project, so they accept our solutions but don't implement them. This indicates that developers' knowledge and implementation of open source license should be enhanced.

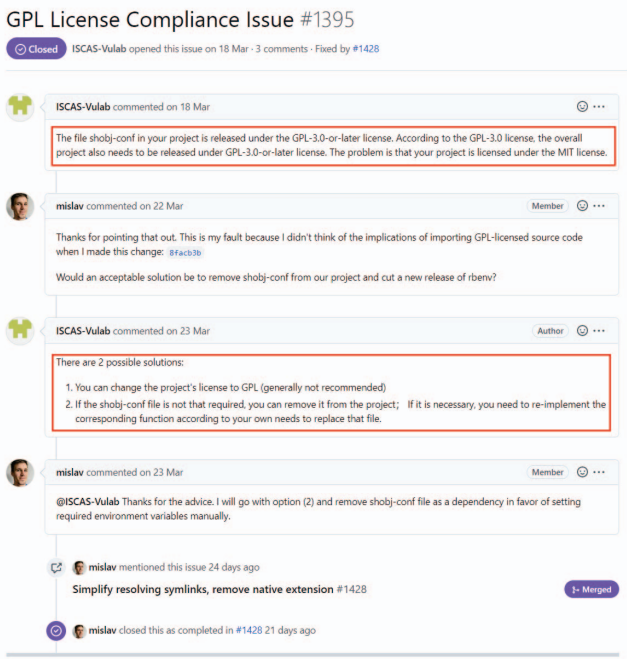

Fig. 4. Regarding a license conflict issue with *rbenv*, we submit comments on GitHub. The developer has since adopted our recommendation and finished the update.

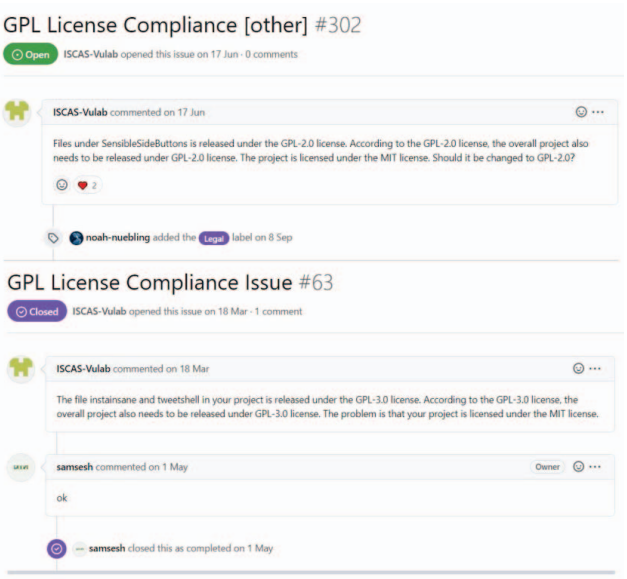

Fig. 5. The developers are interested and receptive to the conflicts we discover but do not bother to resolve them.

VI. THREATS TO VALIDITY

For internal validity, our tool identifies licenses used by files through static detection, but the actual invocation between them is temporarily unavailable. Taking Android as an example, the project declaration uses Apache-2.0, but the Linux Kernel used by Android is licensed under the GPL-2.0. Although the tool detects conflicts, Android uses user space to avoid conflicts in practical use. In addition, some projects use licenses that may have modifications on commonly used terms or new terms added. Our DIKE targets 12 mainstream terms but does not process less commonly used terms. For example, our work does not include Classpath exception 2.0 licenses, which may affect the determination of conflicts.

For external validity, there are other platforms for FOSS. In our work, we only focus on GitHub because it is the most popular one. The projects we detect contain at least 1,000 stars. We believe that this is an acceptable limit because it makes the operational time for conflict detection feasible and because conflicting license behavior has more consequences in an increasingly large and popular project.

VII. RELATED WORK

Open Source License semantic analysis. Analysis of license semantics has been the subject of extensive research. Gangadharan et al. use Open Digital Rights Language to implement the terms of FOSS licenses in a machine interpretable manner [34]. The Qualipso project introduce an ontology that uses Web Ontology Language (OWL) to model software systems. The ontology contains a set of 8 FOSS licenses and their rights [35]. FindOSSLicense classify and summarized license sentences in 24 license through manual analysis and obtained terms to model licenses. Based on a

topic model, FOSS-LTE maps license terms to sentences and sentences to topics using Latent Dirichlet Allocation (LDA) [36]. Numerous studies have been undertaken to analyze license semantics, but the information is typically collected from licensing files by manually examining several licenses, which limits its applicability. Simple topic models may introduce much noise. Our work encompasses a significantly broader range of licenses than earlier work. The approach base on deep learning can extract terms automatically and determine responsibilities with improved performance.

License detection and conflict analysis. Recent research has focused on identifying licenses and analyzing their existence for potential conflicts. FOSSology is a scanning tool for licensing, copyright, and export control scanning that uses the binary Symbol Alignment Matrix method (bSAM) to identify licenses [27]. D.M. German et al. propose Ninka [37], which is sentence-based and gives a simple and efficient way to detect open source licenses in source code files. License Compatibility Checker checks NPM dependency packages for license compatibility based on the SPDX standard according to rules, which provides a straightforward comparison of licenses in packages and explains how much the license permits. LUCE is a solution to license accountability and compliance and can provide a blockchain based solution for the automatic management of data licensing terms and data accountability. It checks compliance process requesting that the data requester periodically confirms in the blockchain network that is complying with the terms [38], [39]. Kapil Singi et al. propose a Compliance Adherence and Governance (CAG) framework using blockchain technologies, which can analyze the events for non-conformant behavior through smart contracts. The framework requires 'compliance policies' to be encoded as smart contracts manually [40]. Despite the advances, past license recognition and conflict analysis studies required a significant amount of a priori knowledge from specialists. In contrast to the previous work, we implemented DIKE, which automates the function of analyzing license terms and responsibilities and builds a license knowledge base, allowing it to explore arbitrary licenses that express flexibility and comprehensively discover potential license conflicts in software.

VIII. Conclusion

For the purpose of creating valuable products in the open source ecosystem, developers and commercial organizations choose FOSS. Maintaining license compliance for FOSS is crucial to ensuring a safe development environment. This emphasizes the significance of making informed decisions about open source licenses and avoiding conflicting projects throughout development. In this paper, we propose DIKE, an automatic and effective tool that identifies license terms and responsibilities, and has a license knowledge base of 3,256 unique licenses. DIKE detects license conflicts for open source software. After detecting a conflict, DIKE can make recommendations to help developers resolve the conflict. We utilize DIKE to conduct an in-depth analysis of numerous Github FOSS projects, looking at: (1) the licenses used in FOSS; (2) whether there are license conflicts with FOSS; and (3) strategies for resolving such conflicts. Our large-scale empirical study on 16,341 projects reveals that 27.2% of the projects are suffering from license conflict. Future efforts will be made to guarantee that licenses are appropriately identified and the license knowledge base can be updated with more data. In the interim, efforts continue to resolve conflicts so that developers may better monitor their licenses and avoid compliance problems.

IX. Acknowledgements

This project is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA0320000), the National Natural Science Foundation of China (No. 62202457), and the project funded by China Postdoctoral Science Foundation (No. 2022M713253).

References

[1] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/libre open-source software development: What we know and what we do not know," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, pp. 1–35, 2008.

[2] F. Mancinelli, J. Boender, R. di Cosmo, J. Vouillon, B. Durak, X. Leroy, and R. Treinen, "Managing the complexity of large free and open source package-based software distributions," in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, 2006, pp. 199–208.

[3] M. Sojer and J. Henkel, "Code reuse in open source software development: Quantitative evidence, drivers, and impediments," *Journal of the Association for Information Systems*, vol. 11, no. 12, p. 2, 2010.

[4] A. Azhakesan and F. Paulisch, "Sharing at scale: an open-source-software-based license compliance ecosystem," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, 2020, pp. 130–131.

[5] S. Chun, S. Yoon, and S. Jeong, "A study on the business model design and economic evaluation of open source software license compliance platform," *Journal of the Korea Society for Simulation*, vol. 29, no. 2, pp. 1–10, 2020.

[6] H. Schoettle, "Open source license compliance-why and how?" *Computer*, vol. 52, no. 8, pp. 63–67, 2019.

[7] A. Mathur, H. Choudhary, P. Vashist, W. Thies, and S. Thilagam, "An empirical study of license violations in open source projects," in *2012 35th Annual IEEE Software Engineering Workshop*. IEEE, 2012, pp. 168–176.

[8] D. A. Almeida, G. C. Murphy, G. Wilson, and M. Hoye, "Do software developers understand open source licenses?" in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 1–11.

[9] T. A. Alspaugh, W. Scacchi, and H. U. Asuncion, "Software licenses in context: The challenge of heterogeneously-licensed systems," *Journal of the Association for Information Systems*, vol. 11, no. 11, p. 2, 2010.

[10] T. Tuunanen, J. Koskinen, and T. Kärkkäinen, "Automated software license analysis," *Automated Software Engineering*, vol. 16, no. 3, pp. 455–490, 2009.

[11] I. Mannino, E. Ninka, M. Turvani, and M. Chertow, "The decline of eco-industrial development in porto marghera, italy," *Journal of Cleaner Production*, vol. 100, pp. 286–296, 2015.

[12] G. M. Kapitsaki and G. Charalambous, "Modeling and recommending open source licenses with findosslicense," *IEEE Transactions on Software Engineering*, vol. 47, no. 5, pp. 919–935, 2019.

[13] R. Gobeille, "The fossology project," in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 47–50.

[14] L. Xiao, X. Huang, B. Chen, and L. Jing, "Label-specific document representation for multi-label text classification," 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)*, 2019, pp. 466–475.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.
[16] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.
[17] T. F. Gordon, "Analyzing open source license compatibility issues with carneades," in *Proceedings of the 13th International Conference on Artificial Intelligence and Law*, 2011, pp. 51–55.
[18] Y. H. Lin, T. M. Ko, T. R. Chuang, and K. J. Lin, "Open source licenses and the creative commons framework: License selection and comparison," *Journal of Information Science & Engineering*, vol. 22, no. 1, p. 2006, 2006.
[19] R. Sen, C. Subramaniam, and M. L. Nelson, "Determinants of the choice of open source software license," *Journal of Management Information Systems*, vol. 25, no. 3, pp. 207–240, 2008.
[20] https://choosealicense.com/appendix/, choosealicense.
[21] R. Sen, C. Subramaniam, and M. L. Nelson, "Open source software licenses: Strong-copyleft, non-copyleft, or somewhere in between?" *Decision support systems*, vol. 52, no. 1, pp. 199–206, 2011.
[22] T. Karopka, H. Schmuhl, and H. Demski, "Free/libre open source software in health care: a review," *Healthcare informatics research*, vol. 20, no. 1, pp. 11–22, 2014.
[23] A. Hemel, K. T. Kalleberg, R. Vermaas, and E. Dolstra, "Finding software license violations through binary code clone detection," in *Working Conference on Mining Software Repositories*, 2011.
[24] H. Zhang, B. Shi, and L. Zhang, "Automatic checking of license compliance," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–3.
[25] V. Del Bianco, L. Lavazza, S. Morasca, and D. Taibi, "Quality of open source software: the qualipso trustworthiness model," in *IFIP International Conference on Open Source Systems*. Springer, 2009, pp. 199–212.
[26] S. Mansfield-Devine, "Open source and the internet of things," *Network Security*, vol. 2018, no. 2, pp. 14–19, 2018.
[27] M. C. Jaeger, O. Fendt, R. Gobeille, M. Huber, J. Najjar, K. Stewart, S. Weber, and A. Wurl, "The fossology project: 10 years of license scanning," *IFOSS L. Rev.*, vol. 9, p. 9, 2017.
[28] G. M. Kapitsaki, F. Kramer, and N. D. Tselikas, "Automating the license compatibility process in open source software with spdx," *Journal of Systems and Software*, vol. 131, pp. 386–401, 2017.
[29] D. Paschalides and G. M. Kapitsaki, "Validate your spdx files for open source license violations," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 1047–1051.
[30] J. Mukhoti, V. Kulharia, A. Sanyal, S. Golodetz, P. H. Torr, and P. K. Dokania, "Calibrating deep neural networks using focal loss," *arXiv preprint arXiv:2002.09437*, 2020.
[31] M. Bicer and X. Zhang, "An efficient, hybrid, double-hash stringmatching algorithm," in *2019 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. IEEE, 2019, pp. 1–5.
[32] X. Wang and D. Pao, "Memory-based architecture for multicharacter aho–corasick string matching," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 143–154, 2017.
[33] S. Hasib, M. Motwani, and A. Saxena, "Importance of aho-corasick string matching algorithm in real world applications," *Journal Of Computer Science And Information Technologies*, vol. 4, pp. 467–469, 2013.
[34] G. Gangadharan, V. D'Andrea, S. De Paoli, and M. Weiss, "Managing license compliance in free and open source software development," *Information Systems Frontiers*, vol. 14, no. 2, pp. 143–154, 2012.
[35] P. Ombredanne, "Free and open source software license compliance: tools for software composition analysis," *IEEE Annals of the History of Computing*, vol. 53, no. 10, pp. 105–109, 2020.
[36] G. M. Kapitsaki and D. Paschalides, "Identifying terms in open source software license texts," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 540–545.
[37] D. M. German, Y. Manabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 437–446.
[38] A. Havelange, M. Dumontier, B. Wouters, J. Linde, D. Townend, A. Riedl, and V. Urovi, "Luce: A blockchain solution for monitoring data license accountability and compliance," *arXiv preprint arXiv:1908.02287*, 2019.
[39] V. Urovi, V. Jaiman, A. Angerer, and M. Dumontier, "Luce: A blockchain-based data sharing platform for monitoring data license accountability and compliance," *arXiv preprint arXiv:2202.11646*, 2022.
[40] K. Singi, V. Kaulgud, R. J. C. Bose, and S. Podder, "Cag: compliance adherence and governance in software delivery using blockchain," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2019, pp. 32–39.