LiDetector: License Incompatibility Detection for Open Source Software

SIHAN XU, TKLNDST, College of Cyber Science, Nankai University, China
YA GAO, TKLNDST, College of Computer Science, Nankai University, China
LINGLING FAN*, TKLNDST, College of Cyber Science, Nankai University, China
ZHELI LIU, TKLNDST, College of Cyber Science, Nankai University, China
YANG LIU, Zhejiang Sci-tech University, China and Nanyang Technological University, Singapore
HUA JI, TKLNDST, College of Cyber Science, Nankai University, China

Open-source software (OSS) licenses dictate the conditions which should be followed to reuse, distribute, and modify software. Apart from widely-used licenses such as the MIT License, developers are also allowed to customize their own licenses (called custom license), whose descriptions are more flexible. The presence of such various licenses imposes challenges to understand licenses and their compatibility. To avoid financial and legal risks, it is essential to ensure license compatibility when integrating third-party packages or reusing code accompanied with licenses. In this work, we propose LiDetector, an effective tool that extracts and interprets OSS licenses (including both official licenses and custom licenses), and detects license incompatibility among these licenses. Specifically, LiDetector introduces a learning-based method to automatically identify meaningful license terms from an arbitrary license, and employs Probabilistic Context-Free Grammar (PCFG) to infer rights and obligations for incompatibility detection. Experiments demonstrate that LiDetector outperforms existing methods with 93.28% precision for term identification, and 91.09% accuracy for right and obligation inference, and can effectively detect incompatibility with 10.06% FP rate and 2.56% FN rate. Furthermore, with LiDetector, our large-scale empirical study on 1,846 projects reveals that 72.91% of the projects are suffering from license incompatibility, including popular ones such as the MIT License and the Apache License. We highlighted lessons learned from perspectives of different stakeholders and made all related data and the replication package publicly available to facilitate follow-up research.

CCS Concepts: • **Software and its engineering** → **Software libraries and repositories**; **Open source model**; **Reusability**.

Additional Key Words and Phrases: Open Source Software, License, Incompatibility Detection

ACM Reference Format:
Sihan Xu, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu, and Hua Ji. 2022. LiDetector: License Incompatibility Detection for Open Source Software . *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2022), 28 pages. https://doi.org/10.1145/3518994

*Lingling Fan is the corresponding author. Email: linglingfan@nankai.edu.cn

Authors' addresses: Sihan Xu, xusihan@nankai.edu.cn, TKLNDST, College of Cyber Science, Nankai University, Tianjin, China; Ya Gao, gaoya_cs@mail.nankai.edu.cn, TKLNDST, College of Computer Science, Nankai University, Tianjin, China; Lingling Fan, linglingfan@nankai.edu.cn, TKLNDST, College of Cyber Science, Nankai University, Tianjin, China; Zheli Liu, liuzheli@nankai.edu.cn, TKLNDST, College of Cyber Science, Nankai University, Tianjin, China; Yang Liu, yangliu@ntu.edu.sg, Zhejiang Sci-tech University, China and Nanyang Technological University, Singapore; Hua Ji, hua.ji@nankai.edu.cn, TKLNDST, College of Cyber Science, Nankai University, Tianjin, China.

1049-331X/2022/1-ART1 $15.00
https://doi.org/10.1145/3518994

1 INTRODUCTION

Open source software (OSS) [40] is a type of software where source code is publicly available under certain licenses. The licenses dictate the conditions under which OSS can be reused, distributed, and modified *legally*. To facilitate software development, a common practice is to integrate OSS code so that developers do not need to reinvent the wheel. While it brings convenience for software development, it also induces security issues [6, 7, 68, 70] and legal risks [34, 44] such as copyright infringement [46], caused by license incompatibility when integrating third-party components. As previous studies [13, 27], *license incompatibility* occurs when there exists no such a license that satisfies all the rights and obligations of all the integrated third-party components, e.g, "MUST disclose source" declared by one license but "CANNOT disclose source" declared by another license within the same project.

According to our preliminary study on 1,846 GitHub projects, 48.86% projects are suffering from license incompatibility. Note that, in the preliminary study, due to the lack of effective tools for incompatibility detection, we only investigate the incompatibility among some popular licenses that can be identified by an existing tool Ninka [15].

To address this issue, there have been several studies that investigate license compatibility, mainly focused on license identification and compatibility analysis [15, 25, 27, 28, 30, 63]. However, there are two problems that limit the application of previous studies. **First**, previous works can only investigate the compatibility between a predefined set of common licenses, and can hardly be adapted to other licenses automatically. For instance, SPDX-VT [27] predefines a dependency graph to tease apart the compatibility relationships specifically for 20 well-known licenses, however, other licenses cannot be addressed by SPDX-VT. **Second**, the rules to detect incompatibility need to be manually defined and specified for each license, which is a major obstacle to automatically detect incompatibility when licenses are changed, updated, or customized by developers. Among previous studies, only FOSS-LTE [28] can be adapted to interpret an arbitrary license. Nevertheless, it can only detect 38 regulations in licenses, and developers need to manually analyze the compatibility for a given project. Actually, popular licenses or official licenses in SPDX [12] often have various versions and exceptions, apart from which, developers are also allowed to create their own licenses (i.e., *custom licenses*). According to our preliminary study, 24.56% license texts are customized by developers. Different licenses regulate different rights and obligations. As a result, it is impractical to identify and manually define the relationships for all licenses in the community. Instead, it is crucial to propose an effective method that automatically interprets licenses and detects license incompatibility issues throughout all kinds of licenses including custom licenses.

To this end, in this paper, we proposed LiDetector, an automated tool for interpreting licenses to detect license incompatibility for open source software. It first constructs a probabilistic model to identify meaningful license terms, and then performs sentiment analysis based on grammar parsing to infer rights and obligations from licenses. Based on the identified terms and the attitudes implied by licenses, LiDetector can identify license incompatibility for arbitrary licenses. Comparative experiments demonstrate the effectiveness of LiDetector, with 93.28% precision and 75.70% recall for license term identification, 91.09% accuracy for right and obligation inference, and 169 incompatible projects identified from 200 Github projects (with 10.06% false positive rate and 2.56% false negative rate). To further investigate license incompatibility in real-word OSS, we leverage LiDetector to conduct an empirical study on 1,846 Github projects and find that 72.91% projects are suffering from license incompatibility, involving some very popular licenses such as the MIT License [57] and the Apache License [53]. In addition, *Disclose Source* induces the most number of conflicts (7,186), which deserves more attention to avoid serious legal risks. Finally, lessons learned are summarized based on our study from the perspectives of different stakeholders (e.g., developers) to shed light on the importance of license compatibility and the usefulness of LiDetector.

In summary, we made the following novel contributions:

- We proposed LiDetector, a hybrid and effective method that automatically understands license texts and infers rights and obligations to detect license incompatibility in open source software with arbitrary licenses, including the widely-used ones and the custom ones.
- Extensive and comparative experiments demonstrate the effectiveness of LiDetector over existing methods, with 10.06% false positive rate and 2.56% false negative rate in license incompatibility detection for open source projects.
- We further conduct a large-scale empirical study on 1,846 GitHub projects by leveraging LiDetector, and find that 72.91% of the projects are suffering from license incompatibility, involving popular licenses such as the MIT and Apache License, which deserve more attention from developers and software companies. We released all the datasets [66] and the replication package [67] on Github for the community.

2 BACKGROUND

In this section, we first introduce OSS licences and the compliance issues. Then, we present a motivating study, which shows the importance of detecting license incompatibility for OSS.

2.1 OSS License

Licenses applied in open source software (OSS) regulate the rights, obligations, and prohibitions of OSS use. An OSS license is represented in the form of text description, where the copyright holders specify the conditions under which users can freely use, modify, and distribute software [38]. The Software Package Data Exchange specification (SPDX) [12, 37] maintains over 400 licenses including very popular ones such as the Apache License, the Academic Free License (AFL), and the GNU General Public License (GPL). When users use OSS to facilitate software development, they are expected to comply with the rights and obligations implicated by the licenses, e.g., *can use*, *cannot redistribute*, and *refuse commercial use*.

License term *vs.* license term entity. In this paper, a *license term* refers to a formal and unified description of the conditions of software use (e.g., *commercial use*), while a *license term entity* refers to a specific expression of a license term in real-world license texts (e.g., *sell* or *offer for sale*). As previous studies [30], there are 23 license terms as displayed in Table 1, each of which represents a type of action that users may do. To better understand these terms and facilitate incompatibility detection, following by the previous studies [25, 28], we further classify the 23 terms into *Rights* and *Obligations*, and consider the conditions of license terms. Nevertheless, there are a variety of licenses (such as official ones and custom ones), leading the expressions of a license term to be various, which impose challenges in identifying license terms from an arbitrary license [25, 27].

Project licenses (PL) *vs.* component licenses (CL). Typically, a software product may contain a *project license*, usually in the form of a LICENSE file in the main directory of the software, which states the conditions of software use. In addition, when incorporating third-party software packages or reusing code [69], licenses that accompany each third-party package or file should also be conformed to. In this paper, to distinguish with the project license, licenses in software other than *project licenses* are called *component licenses*.

Declared licenses *vs.* referenced licenses *vs.* inline licenses. Licenses in OSS products are presented mainly in three forms, i.e., declaration, reference, and text in source code. The *declared licenses* dictate rights and obligations in one or more license files (e.g., LICENSE.txt). Users can capture license information from these files directly without any external resource. The *referenced*

Table 1. License Terms and the Descriptions

Category	No.	Term	Description
Rights	0	Distribute	Distribute original or modified derivative works
	1	Modify	Modify the software and create derivatives
	2	Commercial Use	Use the software for commercial purposes
	3	Relicense	Add other licenses with the software
	4	Hold Liable	Hold the author responsible for subsequent impacts
	5	Use Patent Claims	Practice patent claims of contributors to the code
	6	Sublicense	Incorporate the work into something that has a more restrictive license
	7	Statically Link	The library can be compiled into the program linked at compile time rather than runtime
	8	Private Use	Use or modify software freely without distributing it
	9	Use Trademark	Use contributors' names, trademarks or logos
	10	Place Warranty	Place warranty on the software licensed
Obligations	11	Include Copyright	Retain the copyright notice in all copies or substantial uses of the work.
	12	Include License	Include the full text of license in modified software
	13	Include Notice	Include that NOTICE when you distribute if the library has a NOTICE file with attribution notes
	14	Disclose Source	Disclose your source code when you distribute the software and make the source for the library available
	15	State Changes	State significant changes made to software
	16	Include Original	Distribute copies of the original software or instructions to obtain copies with the software
	17	Give Credit	Give explicit credit or acknowledgement to the author with the software
	18	Rename	Change software name as to not misrepresent them as the original software
	19	Contact Author	Get permission from author or contact the author about the module you are using
	20	Include Install Instructions	Include the installation information necessary to modify and reinstall the software
	21	Compensate for Damages	Compensate the author for any damages cased by your work
	22	Pay Above Use Threshold	Pay the licensor after a certain amount of use

licenses indicate licenses referenced by direct or indirect links, where direct links refer to the license name, version, or the website of the license, and indirect links refer to imported software packages according to which licenses can be found. The detailed information needs to be obtained from external sources, such as pypi [45] (the Python Package Index), SPDX [12, 37], and the hosted pages for OSS licenses. The *inline licenses* refer to license text in the same file of source code, which usually appear on the top of source code files. The *inline licenses* are considered as the most fine-grained licenses, since their scope only cover the source code in the same file with license text.

2.2 License Incompatibility

License incompatibility refers to the conflicts of multiple licenses within the same projects. A license consists of permissive and restrictive statements that specify the requirements for a derivative work. Given a license term, rules associated with it span a range from very permissive ones to highly restrictive ones (i.e., *strong copyleft*). To facilitate software development, developers often need to integrate multiple third-party OSS within one project. For this reason, an open source project may need to comply with more than one licenses. However, there are a variety of licenses and exceptions that regulate different rights and obligations. Moreover, developers are also allowed to create their own licenses (denoted by **custom licenses**), which are more flexible in expressions. The combination of such a variety of licenses often carries out incompatibility issues that prevent correct incorporation of third-party software packages.

Compatibility analysis aims to integrate software components with multiple licenses in a newly-developed software [13]. Therefore, we define *license incompatibility* as follows: two licenses (l_1 and l_2) are incompatible if there exists no such a license that can integrate l_1 and l_2 in a newly-defined

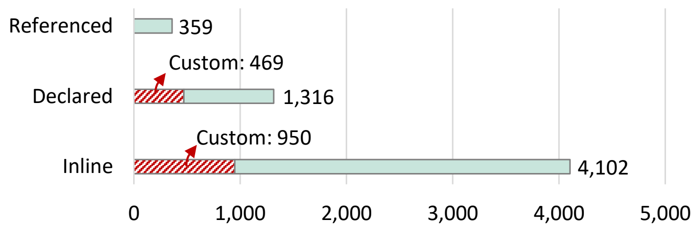


Fig. 1. The Prevalence of Custom Licenses

software without right/obligation conflicts. For instance, if one license declares that *"must contact author for software use"* and another license states *"do not contact authors"*, it indicates that there exists a conflict between the two licenses upon the term *"contact author"*. Developers need to address such a conflict, for instance, choosing another restrictive license to avoid legal risks.

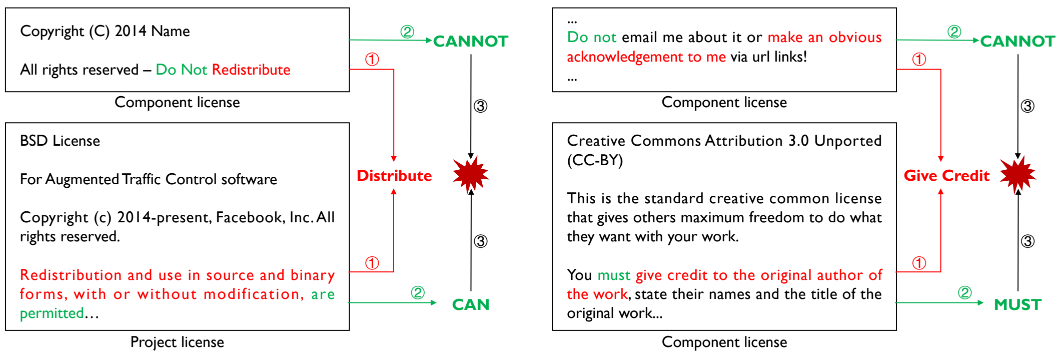
2.3 Motivating Study

To better motivate our work on license incompatibility detection, we conduct an empirical study on real-world open source projects to investigate the prevalence of custom licenses and the incompatibility issues.

The prevalence of custom licenses. We crawled 1,846 popular Python projects ordered by the number of stars from GitHub. For each project, we extract three types of licenses (i.e., the declared, the referenced, and the inline licenses) to investigate the prevalence of custom licenses. Specifically, to extract the declared licenses, we conducted regular matching to identify license files, such as the LICENSE.txt and the COPYING files. To extract the referenced licenses, we obtained license names and versions directly from the project. In addition, for indirectly linked licenses of imported third-party packages, inspired by LicenseFinder [43], we first extracted their names using QDox [42], and then queried pypi [45] (the Python Package Index) by the package names to search for the accompanied licenses. To extract the inline licenses, we extracted license text from comments in the source code files (typically on the top of code files). For extracted license texts, we use Ninka [15], a notable license identification tool, to identify the names and versions of well-known licenses.

As shown in Fig. 1, from 1,846 projects, we obtained 359 unique referenced licenses, 1,316 unique declared licenses, and 4,102 unique inline licenses. In total, we found 5,777 unique licenses. Then, we used Ninka to detect well-known licenses. It was observed that 75.44% of licenses are popular licences that can be detected by Ninka, and 24.56% of the licenses (1,419) are customized by the authors of software products. Note that licenses reported by Ninka may include popular and custom exceptions that are slightly different from the original licenses. Moreover, all the custom licenses belong to the declared and inline licenses. The rational behind is that most referenced licenses are widely-used ones that can be found from external sources by users. Therefore, we further investigated the portion of custom licenses in declared and inline ones, and found that 35.64% of declared licenses and 23.16% of inline licenses are customized licenses. Compared with widely-used licenses, custom ones are more flexible in text. For this reason, although previous works mainly focus on a set of popular licenses, the presence of such a variety of custom licenses requires adaptive methods that are capable of understanding the implications of an arbitrary license.

Incompatible licenses in real-world projects. To investigate the compliance issues of licenses in open source software, we extract the declared, referenced, and inline licenses from the 1,846 projects using the aforementioned extraction method. Note that to observe the prevalence of licence incompatibility, in this motivating study, we only focus on widely-used licenses that can


Copyright (C) 2014 Name
All rights reserved – Do Not Redistribute
Component license
BSD License
For Augmented Traffic Control software
Copyright (c) 2014-present, Facebook, Inc. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted…
Project license
① ② ③
CANNOT
Distribute
CAN
…
Do not email me about it or make an obvious acknowledgement to me via url links!
…
Component license
Creative Commons Attribution 3.0 Unported (CC-BY)
This is the standard creative common license that gives others maximum freedom to do what they want with your work.
You must give credit to the original author of the work, state their names and the title of the original work...
Component license
CANNOT
Give Credit
MUST

(a) Incompatibility between PL and CL in *Augmented Traffic Control* [10]

(b) Incompatibility between CL and CL in *Faust* [49]

Fig. 2. Two Running Examples

be identified by Ninka and have labels on tldrlegal [30], a platform that provides the rights and obligations towards license terms (e.g., redistribute, modify) for well-known licenses.

With the assistance of tldrlegal, we conduct an investigation towards the incompatibility issues in real-world OSS. The result shows that about **48.86%** of the projects suffer from license incompatibility issues. To avoid involving illegal issues, authors of these projects need to analyze component licenses and address the incompatibility issues before distributing their software products.

2.4 Running Example

Fig. 2 depicts two real-world running examples of license incompatibility, where PL denotes a project license, and CL denotes a component license. The first project is Augmented Traffic Control [10], an open source project to simulate network traffic. It contains a license file for the whole project (i.e., a project license), which is an official license named BSD License [54]. Meanwhile, the project also contains a component of atc cookbooks, where a custom license can be found. The project license states that "*Redistribution and use in source and binary forms, with or without modification, are permitted*". However, the component license declares that "*Do Not Redistribute*". It can be seen that the two licenses convey different attitudes towards the same license term (i.e., Distribute in Table 1). Users who comply with the project license (CAN distribute) may still violate the component license (CANNOT distribute). For this reason, we say there exists license incompatibility between the project and component licenses.

Fig. 2b illustrates another example from Faust [49], a Python stream processing library. The project contains two component licences, one of which is a custom license that states "*Do not email me about it or make an obvious acknowledgement to me via url links*". Another component license is an official license named Creative Commons Attribution 3.0 Unported [59], where the authors declare that "*You must give credit to the original author of the work*". In this project, two component licenses convey conflict attitudes towards the same license term (i.e., CANNOT give credit *vs.* MUST give credit). Since one can not develop a new license to satisfy the two component licenses simultaneously, we say there exists license incompatibility between these component licenses. Such license incompatibility can represent a serious threat to the legal use of OSS.

2.5 Problem Statement

In this paper, given a project, we address all types of licenses involved in the project, including the inline, the declared, and the referenced licenses. The collected licenses usually include a project

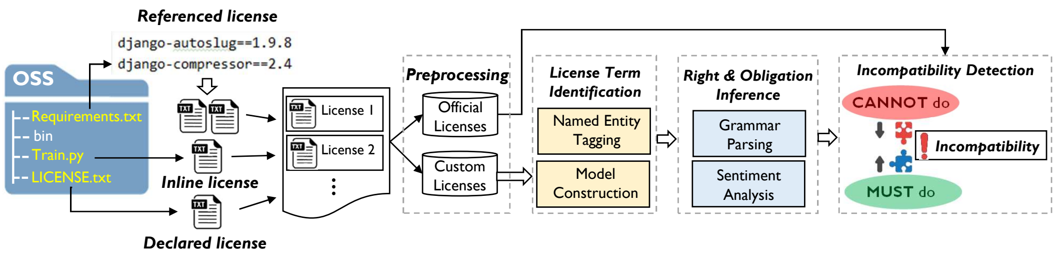


Fig. 3. Overview of LiDetector

license PL and n component licences, i.e., $\{CL_1, CL_2, CL_3, ..., CL_n\}$. Each license declares rights and obligations that users should comply with. Our problem is to check whether there exists license incompatibility (i.e., with conflict rights or obligations) among all kinds of licenses, with consideration for (1) the differences between project licenses and component licenses, i.e., pl *vs.* cl_i and cl_i *vs.* cl_j, where i and j represent the i^{th} and j^{th} component license, respectively; (2) the diversity of licenses; and (3) condition constraints between license terms.

3 APPROACH

3.1 Overview

This section details our approach, LiDetector, a hybrid method that automatically understands license texts and infers rights and obligations to detect license incompatibility in open source software. As depicted in Fig. 3, given an open source project, we first extract three types of licenses, i.e., the referenced, the inline, and the declared licenses, for further incompatibility analysis. After obtaining the set of licenses, the main components of LiDetector include: (1) *Preprocessing*, which filters out official licenses and feeds custom ones into the probabilistic model for automatic understandings of license texts; (2) *License term identification*, which aims to identify the license terms (as displayed in Table 1) relevant to rights and obligations; (3) *Right and obligation inference*, which infers the stated condition of software use defined by license terms; (4) *Incompatibility detection*, which automatically analyzes incompatibility between multiple licenses within one project based on the regulations inferred from each license. We detail each phase as follows.

3.2 Preprocessing

Given a project, after obtaining all the license texts, we filter out official licenses whose rights and obligations have already been known, and feed the other licenses such as newly-defined official licenses and custom licenses into the probabilistic model for automatic understanding of license texts. Here, official licenses denote licenses from the Software Package Data Exchange (SPDX) [12]. To filter out official licenses, we note that only license texts that exactly match the listed official licenses are marked as official licenses. For license texts that contain or reference an official license, we extract the official license and then feed the rest texts into the machine learning model to infer additional rules. Then, given license texts, we remove non-textual parts, check spellings, perform stemming and morphological to obtain the roots of tokens. We utilize the Natural Language Toolkit [39] to preprocess license texts.

3.3 License Term Identification

In this section, we elaborate the method to identify license terms related to rights and obligations.

3.3.1 ***Named Entity Tagging***. Since license texts are typically long and complicated, it is not easy to directly interpret license texts. For each license, LiDetector first identifies *named entities*

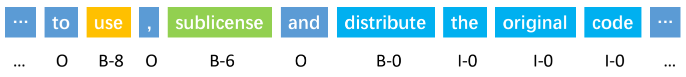

… to use , sublicense and distribute the original code …
… O B-8 O B-6 O B-0 I-0 I-0 I-0 …

Fig. 4. Illustration of the BIO Labelling Mode for Licenses

of license terms which state the conditions of software use. In this paper, a *named entity* refers to a specific expression of a *license term* shown in Table 1, which can be a word, a phrase, or a sentence. Inspired by Named Entity Recognition (NER) [11, 21, 22] in natural language processing, this paper utilizes sequence labeling to identify and localize license term entities. Before sequence labeling, LiDetector first splits each license text into sentences by *Stanford CoreNLP* [20], an integrated framework for natural language processing. After that, we employ the BIO (Begin, Inside, and Outside) mode [47] to label each token in a sentence. As illustrated in Fig. 4, *B-X* implies that the current token is at the beginning of a named entity of the X^{th} license term in Table 1. For instance, *B-0* in Fig. 4 implies that *distribute* is the beginning token of a named entity whose license term is the first one in Table 1. Similarly, *I-X* implies that the current token is inside a named entity of the X^{th} license term, and *O* represents a token outside name entities. In the training phase, we manually tag each sentence by the BIO mode, and obtain a training dataset for sequence labelling of license sentences. In the inference phase, LiDetector automatically predicts the labels of tokens in license texts, so as to identify and localize license term entities.

3.3.2 ***Model Construction.*** Based on the tagged dataset, we train a probabilistic model to predict the label of each token in a license text. As illustrated in Fig. 5, the model consists of three parts: word embedding, sentence representation, and probability calibration. (1) *Word embedding.* To serve as the input of the model, we first embed words in license sentences into vectors. Since licenses are expressed by natural language, we exploit prior knowledge on word semantics and employ the pre-trained Glove model [48] for word embedding. (2) *Sentence representation.* To represent each sentence in the license text, we feed the results of word embeddings into a bi-Directional Long Short-Term Memory (bi-LSTM) [64] model, and learn the representation of each license sentence. (3) *Probability calibration.* Given a token and its context vector learned by bi-LSTM, the model then calibrates its probability distribution over each category (label) by Conditional Random Fields (CRF) [3], so that the contextual information represented by the hidden layer state can be utilized to make a global decision. To reduce the labelling effort and enhance performance, we implement the probabilistic model by semi-supervised training. Specifically, after training with labelled samples, we used the trained model to predict license terms for unlabeled samples. Then, all samples, including labelled samples and other samples with pseudo labels, were collected to train the model. The rationale behind is that pseudo labelling of unlabelled samples are also predictive. Finally, the output of the probabilistic model is a sequence of labels from where license term entities can be directly inferred.

For instance, given the first running example in Section 2.4, the label sequence of the sentence *"Redistribution and use in source and binary forms, with or without modification, are permitted"* predicted by the probabilistic model is *{B-0, I-0, I-0, I-0, I-0, I-0, I-0, I-0, O, O, O, O, O, O}*. By this means, LiDetector can localize the license term entity *"redistribution and use in source and binary forms"*, and identify the license term "Distribute" from the sentence.

3.4 Right and Obligation Inference

After localizing license terms from texts, LiDetector infers the attitudes of originators towards these license terms (e.g., grant/reserve certain rights), which are the rights and obligations stated by the license. LiDetector achieves this goal by three steps. First, it parses sentences where license terms are localized. Then, based on the grammar parsing, it identifies tokens which convey

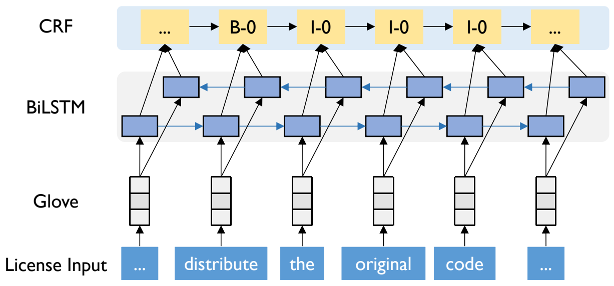


Fig. 5. The Probabilistic Model

permissive or restrictive attitudes towards license terms. Finally, it infers the relationships between identified license terms, since some license terms can be the conditions of other terms.

3.4.1 ***Grammar Parsing.*** To infer the conveyed attitudes towards identified license terms, we first conduct grammatical analysis on the sentences where term entities are localized. Since license texts are expressed by natural language under universe grammar rules, we exploited the pretrained Probabilistic Context-Free Grammar (PCFG) model [50] from *Standford CoreNLP* [20] to assign a probability to each grammar rule. By this means, a syntax tree can be obtained by selecting the tree with the highest probability scores. Fig. 6 displays an example of grammar parsing on the sentence *"You can not refuse such a promise that significant changes must be declared"* from CC-BY-SA-4.0 [55]. In this case, *"significant changes must be declared"* is an identified term entity, which refers to a license term "State Changes" (No.15 in Table 1). In Fig. 6, non-terminal nodes such as *NP* and *VB* represent the part-of-speech tags [35], and a leaf node such as *refuse* and *changes* represents a token in the license sentence. Table 2 shows the part-of-speech tags associated with their descriptions. By traversing the grammar tree, we can obtain the full path of each leaf node, which is the grammatical sequence of the token in the parsed sentence. For example, from Fig. 6 we can acquire the path from the *ROOT* node to the leaf node *refuse* as $ROOT \longrightarrow S \longrightarrow VP \longrightarrow VP \longrightarrow VB \longrightarrow refuse$. It indicates that *refuse* is a verb in the verb phrase *"refuse such a promise that significant changes must be declared"* (denoted by V_1), which is dominated by *can* and *not*, two tokens in a larger verb phase that contains V_1. Taking Fig. 2a as an example, it can be inferred that the license term entity *"redistribution and use in source and binary forms"* is a noun phrase (NP) in the sentence, and the attitude towards this license term is affected by the verb phrase (VP) *"are permitted"*.

3.4.2 ***Sentiment Analysis.*** Based on the results of license term identification and grammar parsing, we perform sentiment analysis to infer the attitudes towards these terms. Generally, the attitudes towards license terms can be categorized into three types, i.e., *MUST*, *CANNOT*, and *CAN*. Given a target sentence l and a license term k identified from the sentence, LiDetector infers the attitude towards k via sentiment analysis. When the attitudes towards all identified terms are inferred, LiDetector obtains a summary of rights and obligations of the whole license, i.e., $T(l) = [t_0, t_1, ..., t_{22}]$, where t_i represents the attitude towards the i^{th} term in Table 1, $t_i \in \{\text{CAN, CANNOT, MUST, UNKNOWN}\}$, and $0 \leq i \leq 22$. Note that absent license terms are marked with UNKNOWN. Specifically, we first define a set of parts of speech that may convey permissive or restrictive attitudes of authors, i.e., Verb (VB, VBD, VBG, VBN, VBP, VBZ) and Others (MD, IN, RB, RBR, RBS). Tokens with these parts of speech are regarded as *powerful tokens* (PTs). For instance, the token *you* in Fig. 6 is not a powerful token since its part of speech is RPR, which is supposed to have no influence on the attitudes. After obtaining all PTs, we further divide them into two groups according to their relationships with the target entity.

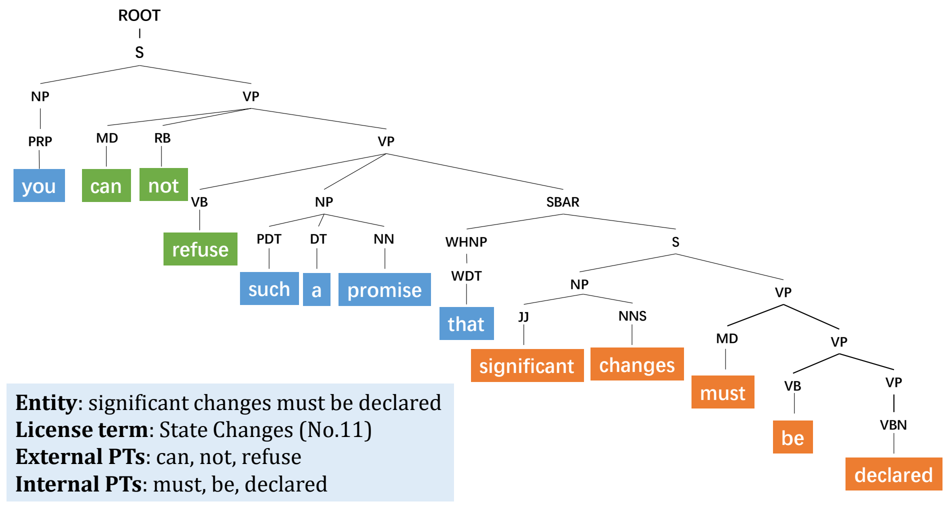

ROOT
S
NP
VP
PRP
MD
RB
VP
you
can
not
VB
NP
SBAR
refuse
PDT
DT
NN
WHNP
S
such
a
promise
WDT
that
NP
VP
JJ
NNS
significant
changes
MD
VP
must
VB
VP
be
VBN
declared
Entity: significant changes must be declared
License term: State Changes (No.11)
External PTs: can, not, refuse
Internal PTs: must, be, declared

Fig. 6. The Parsing Result of a Simplified Sentence in CC-BY-SA-4.0 [55]

- **Internal PTs**: An internal PT represents a token contained in the target entity. Typically, internal PTs directly declare the rights and obligations towards the target license term. For example, in Fig. 6, the tokens *must*, *be*, and *declared* are three internal PTs of the entity *significant changes made to software must be declared within the copies.*
- **External PTs**: As a part of a sentence, the sentiment analysis towards the attitudes needs to consider external PTs which are outside the target entity but have a dominant influence on the attitude towards the target entity. For example, in Fig. 6, tokens such as *can*, *not*, and *refuse* are three external PTs that dominates the attitude of the entire sentence.

Table 2. Partial List of the Part-of-speech Tags and Description, more Tags are Described in [35]

Tag	Description	Tag	Description
ROOT	Text to process	VP	Verb phrase
S	Simple declarative clause	SBAR	Clause by a subordinating conjunction
NP	Noun phrase	WHNP	Wh-noun Phrase
VB	Verb, base form	MD	Modal
VBD	Verb, past tense	IN	Preposition or subordinating conjunction
VBG	Verb, gerund or present participle	RB	Adverb
VBP	Verb, non-3rd person singular present	RBR	Adverb, comparative
VBZ	Verb, 3rd person singular present	RBS	Adverb, superlative
VBN	Verb, past participle	PRP	Personal pronoun
PDT	Predeterminer	NN	Noun, singular or mass
DT	Determiner	NNS	Noun, plural
JJ	Adjective	WDT	Wh-determiner

Given a license term entity, we first collect its internal PTs that directly declare the rights and obligations. Then, based on the parsing results, we search for the external PTs that may dominate the attitudes towards the target entity. By this means, we can acquire a set of PTs that are further used to infer the rights and obligations implied by licenses. Since the expressions of attitudes are not as flexible as those of license terms, we define a set of words and phrases in Table 3 as the expressions of each attitude. Finally, for each PT, we apply a heuristic strategy to infer the attitudes. Specifically, we mark a PT with CANNOT or MUST if it belongs to the corresponding expressions listed in Table 3; otherwise, we mark it with CAN. Double CANNOT is offset.

Table 3. Expressions of Attitudes

Attitude	Expression
CAN	–
CANNOT	not, without, notwithstand, refuse, disallow, decline, against, delete, nor, void, neither, prohibit, remove, don't, no, nothing
MUST	must, should, as long as, so long as, shall, provided that, ensure that,ask that, have to

We note that for right-related terms, as previous studies [8], the absence of them are marked with CANNOT since no rights are explicitly granted in the license; while for obligation-related terms, the absence of them are marked with CAN. In this way, we can infer the attitude towards each license term. For example, in Fig. 2a, there are no internal PTs in the license term entity *"redistribution and use in source and binary forms"*, and the external PTs are {*"are"*, *"permitted"*}. According to Table 3, LiDetector infers the attitude towards the license term *"Distribute"* as CAN, and thus the right conveyed by the project license is *"CAN Distribute"*.

3.4.3 ***Condition Relationship.*** After identifying license terms and the attitudes towards them, rights and obligations related to each term can be inferred (e.g., CANNOT Redistribute). Although we treat each license term independently in the previous steps, license terms can also be the conditions of other terms. For instance, *"you can modify if you state changes"*. In this case, there are two license terms (i.e., "Modify" and "State Changes"), and the right (i.e., "Modify") is only granted under certain constraint (i.e., "State Changes"). To address this issue, we analyze the condition relationships between license terms. Specifically, based on the parsing results, we identify the conditional clauses that state the conditions of software use. License terms in the conditional clauses are the conditions of terms in the main clause. Finally, we separately assume the condition is satisfied or not, and update the attitudes of license terms in both conditional and main clause. We describe incompatibility analysis when faced with conditions in Section 3.5.

3.5 Incompatibility Detection

As previous studies [13, 27], we define license compatibility as the ability to combine multiple licenses into the same software product. Some incompatible examples are aforementioned in the running examples. Based on the rights and obligations implied by each license, it is still challenging to accurately detect license incompatibility due to: (1) project licenses (PL) should be more restrictive than component licenses (CL), i.e., PL and CL should be treated differently. (2) Some rights or obligations are only granted under certain conditions.

To address the first challenge, given a target project, LiDetector discriminates between project and component licenses, and define license compatibility rules as follows:

- **Rule 1: Compatibility between component licenses**. Two component licenses CL_1 and CL_2 are compatible if it is possible to develop a new license L that anyone who conforms to license L will not violate license CL_1 and CL_2.
- **Rule 2: Compatibility between a project license and a component license**. A project license PL is "one-way compatible" with a component license CL, if anyone who conforms to license PL will not violate license CL.

Based on the definitions, for each pair of licenses, we compare their attitudes towards the same license term one by one. As illustrated in Table 4, a project license PL is "one-way-compatible" with a component license CL, if PL is the same or more restrictive than CL. For instance, MUST and CANNOT are stricter than CAN, so that anyone who conforms to MUST or CANNOT will not

Algorithm 1: License Incompatibility Detection

```
Input: : $l_1 < t_1, atti_{t1} >$ and $l_2 < t_2, atti_{t2} >$: A pair of licenses with extracted terms and attitudes
Output: $Incompatible$ or $Compatible$
1  if $l_1.hasCondi() \vee l_2.hasCondi()$ then
       // if $l_1$ or $l_2$ has conditions, check compatibility when the conditions are satisfied or not.
2      $R$ = condiCheck($l_1$, $l_2$)
3      foreach $(r1, r2) \in R$ do
4          if $(l_1.isCL \wedge l_2.isCL) \wedge \neg (r_1 \vee r_2)$ then
               // when $l_1$ and $l_2$ are both component licenses, under both conditions the licenses are incompatible.
5              return $Incompatible$
6          if $(l_1.isCL \wedge l_2.isPL) \wedge \neg (r_1 \wedge r_2)$ then
               // when $l_1$ is a component license and $l_2$ is a project license, at least under one condition the licenses are
               incompatible.
7              return $Incompatible$
8      return $Compatible$
9  else
10     if $\neg checkIncomp\ (l_1, l_2)$ then
           // For terms without conditions, use Rule1&Rule2 to check compatibility for each term.
11         return $Incompatible$
12     else
13         return $Compatible$
14 Function condiCheck($l_1$, $l_2$):
15     $R < r_1, r_2 > \leftarrow \emptyset$ // Incompatibility detection result pairs when considering the path conditions
16     $\mathcal{T} \leftarrow t_{condi}(l_1, l_2)$ // Terms with conditions
17     foreach $term \in \mathcal{T}$ do
           // Assume the condition is True, update the attitudes of the related terms, and detect incompatibility.
18         $condi \leftarrow TRUE$
19         $L < l_1', l_2' > \leftarrow$ UpdateAtti($term$, $l_1$, $l_2$, $condi$)
20         $r_{true} \leftarrow$ checkIncomp ($l_1'$, $l_2'$)
           // Assume the condition is False
21         $condi \leftarrow FALSE$
22         $L < l_1', l_2' > \leftarrow$ UpdateAtti($term$, $l_1$, $l_2$, $condi$)
23         $r_{false} \leftarrow$ checkIncomp ($l_1'$, $l_2'$)
24         $R \leftarrow R \bigcup < r_{true}, r_{false} >$
25     return $R$
```

violate CAN. For a pair of *CLs*, as shown in Table 5, they are compatible with each other only when they can be incorporated into the same software product, so only CANNOT and MUST towards the same license term (e.g., *distribute*) are regarded as incompatible attitudes in this case. Finally, we note that as declared by *choosealicense* [8], the absence of a license implies that nobody can copy, distribute, or modify the work. Therefore, if a project is **without a PL**, we consider all rights are reserved, so the PL is the most restrictive license that are compatible with any CL in the same project. In this case, we only check compatibility among CLs. Similarly, if a license term is not mentioned in the license, all right-related terms are set to CANNOT, and all obligation-related terms are set to CAN by default. For instance, if the license text does not mention anything about redistribution, then it means nobody can redistribute the work.

For example, in Fig. 2a, LiDetector infers that the project license declares "CAN Distribute", while the component license declares "CANNOT Distribute". Since the project license is more

Table 4. Compatibility between PL and CL

PL \ CL	CAN	CANNOT	MUST
CAN	✓	✗	✗
CANNOT	✓	✓	✗
MUST	✓	✗	✓

Table 5. Compatibility between CL_1 and CL_2

CL_1 \ CL_2	CAN	CANNOT	MUST
CAN	✓	✓	✓
CANNOT	✓	✓	✗
MUST	✓	✗	✓

permissive than the component license upon the same license term, LiDetector detects incompatibility between two licenses. For the second example in Fig. 2b, LiDetector infers that the first component license conveys "CANNOT give credit", while the second component license states "MUST give credit". Since it is impossible to comply with both of them simultaneously, LiDetector infers there exists incompatibility between two component licenses.

To address the second challenge, we consider both conditional cases separately when handling terms with conditions, i.e., separately assume the condition is True or False to detect potential incompatibility issues. In this way, we can eliminate the effect of conditions, and employ the above incompatibility checking rules for both cases individually.

Algorithm 1 details the detection process. Specifically, it takes as input a pair of licenses within a projects (l_1, l_2), with extracted terms (t_1, t_2) and the associated attitudes ($atti_{t1}$, $atti_{t2}$), and outputs whether there exist license incompatibility issues. For l_1 and l_2, if at least one of them has terms with conditions, the method `condiCheck()` is invoked to obtain the results under both conditional cases (Lines 1-2 and Lines 14-25), otherwise, we directly invoke the `checkIncomp()` method to detect incompatibility (Lines 10-13). Here, the `checkIncomp()` method detects incompatibility using Rule1 and Rule2 described above. As for the `condiCheck()`, we initialize a list $R < r_1, r_2 >$ to store all the result pairs of both conditional cases, and extract all the terms $\mathcal{T}$ with conditions (Lines 15-16). For each $term$ in $\mathcal{T}$, we first assume the condition is $True$, update the attitudes of the corresponding terms, and check the incompatibility (result stored in r_{true}) (Lines 18-20); then we assume the condition is $False$, update the attitudes of the related terms again, check incompatibility in this case, and the checking result is stored in r_{false} (Lines 18-20). For instance, given a conditional term *"CAN modify if you state changes"*, we split it into two cases: (1) MUST state changes & CAN modify (2) CANNOT modify. Then, we separately check license incompatibility for both cases and obtain the detection results $< r_{true}, r_{false} >$ for both cases. Finally, the results of both cases are stored in R and returned (Lines 24-25). For each result pair in R, we treat PL and CL differently. Specifically, if l_1 and l_2 are both CL, according to Rule 1, they are regarded as *incompatible* only when both results show incompatibility (Lines 4-5). If l_1 or l_2 is a PL, according to Rule 2, they are regarded as *incompatible* if there exists at least one case showing incompatible (Lines 6-7).

4 EVALUATION

In this section, we present the evaluation results of LiDetector to show that it can efficiently detect license incompatibility for open source software. Specifically, we first present the performance of LiDetector in two phases (license term identification and attitude inference), and then demonstrate the effectiveness of LiDetector to detect incompatibility compared with the state-of-the-art tools.

4.1 Preparation

4.1.1 ***Data Preparation.*** The evaluation was conducted in three phases (i.e., license term identification, right and obligation inference, and the overall incompatibility detection). In the first two phases, LiDetector performed tagging, training, and testing in **sentences**. Specifically, we collected the license sentences from two sources: (1) **tldrlegal** [30], a platform where licenses can be uploaded, summarized, and managed by users. We collected license sentences accompanied with license term tags (i.e., CAN, CANNOT, and MUST), and obtained 11,973 labeled sentences from 212 labelled licenses on *tldrlegal.* (2) **Github**. We crawled 1,846 popular projects with more

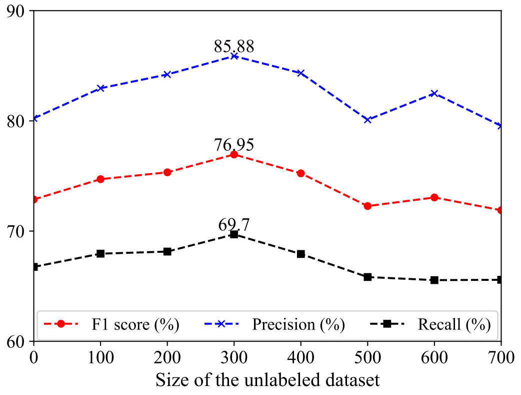


Fig. 7. Unlabeled Data Size Determination for Semi-supervised Training of LiDetector

than 1,000 stars and extracted licenses from them. Then, we filtered out duplicated licenses and finally obtained 48,275 sentences from GitHub. Then, we randomly selected 9,871 sentences (i.e., 20%) to carefully label, and the remaining 38,404 sentences from 754 unlabelled licenses were fed into the semi-supervised learning to enhance the performance of identifying license terms. In total, we obtained 21,844 labeled sentences (i.e., 11,973 from *tldrlegal* and 9,871 from Github). We randomly split these samples into the training and testing datasets by 4:1, and further split the training dataset for training and validation by 4:1. Therefore, in the first two phases, the training, validation, and testing dataset consists of 13,980, 3,495, and 4,369 license sentences. Three authors cross-validated the labels and a lawyer from Yingke Law Firm[1] was involved in the validation. All the techniques and tools were evaluated on the same testing dataset. We have made the dataset publicly available [66].

4.1.2 ***Parameter Settings for Semi-Supervised Learning.*** To save the manual efforts of labelling licenses, we employ a semi-supervised learning method to identify license terms, so that both labelled and unlabelled licenses can be utilized to train the probabilistic model. The rationale behind is that pseudo labels predicted by the model are also predictive and may benefit the model performance when involved in training. Before evaluating the performance of LiDetector, we first investigate how the number of unlabelled samples influences the performance of LiDetector. Specifically, we randomly select a set of numbers of licenses (i.e., 100, 200, 300, 400, 500, 600, 700) from the 754 unlabelled licenses, extract the sentences, and add them into the training dataset respectively. We use three metrics to evaluate the performance of LiDetector, i.e., precision ($P = \frac{TP}{TP+FP}$), recall ($R = \frac{TP}{TP+FN}$), and F1 score ($F1 = \frac{2*P*R}{P+R}$). To avoid bias, for each parameter, we randomly select the same number of unlabelled licenses for three times, and average the results. Fig. 7 shows the performances of LiDetector accompanied with different sizes of unlabelled licenses. It can be observed that the performance of LiDetector peaked when 300 unlabeled licenses were added for semi-supervised training. In the Fig. 7, it can be observed that when more unlabeled data were involved in the semi-supervised training, the performance of LiDetector first increased and reached a peak when the size of unlabeled data is 300. It is consistent with the hypothesis that pseudo labels are predicted for the unlabeled data. However, when the size of unlabeled data is larger than 300, the performance of LiDetector decreased. A possible reason could be that pseudo labels might introduce some noisy data, which prohibited the enhancement of model performance. Therefore, to achieve the best performance, we decide to add 300 unlabeled licenses (i.e., **15,312** sentences) into the training dataset for our semi-supervised learning phase in

[1]www.yingkeinternational.com

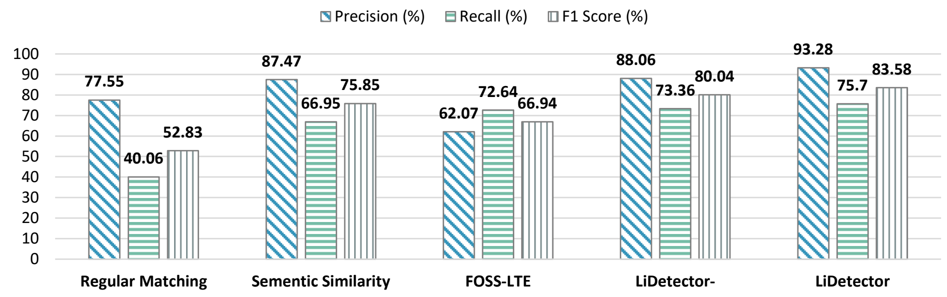


Fig. 8. Comparison on Term Identification

LiDetector in the following experiments. All experiments were conducted on a machine with Intel (R) Core (TM) i5-7200U CPU @ 2.70 GHz and 4.00 GB RAM.

4.2 Evaluation on License Term Identification

4.2.1 ***Setup***. To evaluate the ability of LiDetector to identify license terms, we compare it with FOSS-LTE [28], a state-of-the-art tool that extracts license terms from texts, and two natural language processing (NLP) techniques, i.e., regular matching [3] and semantic similarity [31]. Moreover, to study the influence of unlabelled training samples, we also conduct an ablation study, where LiDetector trained without unlabeled training samples is denoted by LiDetector$^-$. To conduct a fair comparison, we carefully implement the following NLP techniques and adapt them for license term identification. Specifically,

- **Regular Matching** [3], which predefines a set of keyword patterns to guide license term identification. To implement this strategy, we manually analyzed license texts to find as more expressions of license terms as possible. Finally, 72 patterns were found for 23 license terms, as listed on our website [66]. We then use CoreNLP [20] to split licenses into sentences and search for predefined expressions of license terms by regular matching.
- **Semantic Similarity** [29], which utilizes doc2vec [31] to represent text as a vector and searchs for similar expressions of license terms. To adapt this method, we trained the doc2vec model on the aforementioned 754 unlabeled licenses (i.e., 38,404 sentences), so as to learn the representations of license sentences. After that, we manually analyzed license sentences and collected a set of representative sentences for each license term. In total, we collected 51 representative sentences for 23 license terms, as listed on our website [66]. Finally, given a license sentence, we use the trained doc2vec model to predict its representation. Sentences with similar representations are considered to convey similar regulations about software use.

We conduct the comparative studies on the ground-truth dataset described in Section 4.1 (i.e., 400 labelled licenses with 21,844 labeled sentences). We randomly split these samples into the training and testing datasets by 4:1, and further split the training dataset for training and validation by 4:1. All the techniques and tools are evaluated on the same testing dataset.

4.2.2 ***Results***. Fig. 8 reports the results of each method. It can be seen that LiDetector outperforms the other methods, achieving 83.58% F1-score, followed by LiDetector$^-$ (80.04%) and semantic similarity (75.85%). Among five methods, regular matching has the worst performance with 52.83% F1 score. The reason could be that predefined patterns limit the flexibility of expressions, leading to a low recall. Compared with recall, the precision of regular matching is relatively high due to the strict matching strategy. It can also be seen that semantic similarity outperforms regular matching and FOSS-LTE. The results indicate that semantics of license sentences can be learned by the doc2vec model, so that sentences with similar representations contain similar license terms. Moreover, we can also observe that the precision of FOSS-LTE is the lowest among all five methods. A possible reason could be that as a topic model, FOSS-LTE may be affected by noisy data during unsupervised learning.

Compared with FOSS-LTE, regular matching, and semantic similarity, both LiDetector and LiDetector$^-$ have higher precision, i.e., 93.28% and 88.06%, respectively. The reasons behind are two folds: (1) the proposed method filters out license sentences which are irrelevant to license terms by preprocessing and clustering; (2) we employ a semi-supervised learning method that trains a sequential model on labelled sequences, which may lead to higher precision compared with unsupervised learning methods such as topic model in FOSS-LTE. From the ablation study, it can be seen from Fig. 8 that LiDetector (semi-supervised learning with unlabelled samples) outperforms LiDetector$^-$ (supervised learning without unlabelled samples) for both precision and recall. The results indicate that pseudo labelling of unlabelled samples are predictive, and it is worthy of incorporating these unlabelled samples for training.

However, there are also corner cases that LiDetector fails to identify license terms. For example, in the project Statsite [61], the license sentence *"You may add your own copyright statement to your modifications and may provide additional or different license terms and conditions for use ...."* actually grant rights to Relicense (No.3 in Table 1). However, LiDetector failed to identify this term due to the rareness of such expressions in the training dataset.

4.3 Evaluation on Right and Obligation Inference

4.3.1 ***Setup.*** After identifying license terms, LiDetector aims to infer the attitudes towards these terms from license texts. To conduct a comprehensive study, in addition to FOSS-LTE [28], we also compare LiDetector with two NLP techniques, i.e., regular matching [52] and SST-based sentiment analysis [51]. Again, these NLP techniques were implemented and adapted to the context of licenses, so as to infer rights and obligations.

- **Regular matching** [52], which predefines a set of keywords to infer the attitudes implied by license sentences. To conduct a fair comparison, we use the same set of keywords (as listed in Table 3) for the baseline and LiDetector. Specifically, the baseline searches for the keywords that represent attitudes in the order of CANNOT, MUST, and CAN, which denote the prohibition, obligation, and right of a software product, respectively.
- **SST-based sentiment analysis** [51], which learns a classifier that predicts the positive or negative attitudes from the Stanford Sentiment Treebank (SST). Since licenses are written in natural language, we train a LSTM model based on the SST sentiment dataset, so as to predict the attitudes of authors behind licenses.

To investigate the effectiveness of LiDetector, we conduct a comparative study on the ground-truth dataset (i.e., 400 labelled licenses with 21,844 sentences), and randomly split the dataset into training, validation, and testing datasets as described in Section 4.2. All the methods are evaluated on the same testing dataset. Moreover, to conduct a fair comparison, we evaluate the effectiveness of these methods on the same set of license terms, which has been correctly identified in the previous phase. For this reason, here we only compare the *accuracy* of these methods, since there are no false negatives in the evaluation and the *recall* cannot be calculated.

4.3.2 ***Results.*** Table 6 shows the results on right and obligation inference. It can be observed that LiDetector outperforms the other methods in attitude inference, achieving 91.09% accuracy. SST-based sentiment analysis achieves a comparable performance with FOSS-LTE and regular matching, with the accuracy around 82%. Note that we utilize a model that has been well trained over the SST sentiment dataset. However, the results reported in Table 6 indicate that although

Table 6. Comparison on Right and Obligation Inference

Method	Accuracy (%)
Regular Matching	81.27
SST-based Sentiment Analysis	82.88
FOSS-LTE	82.71
LiDetector	**91.09**

Table 7. Evaluation Results for License Comprehension

Term Extraction	R. and O. Inference	P (%)	R (%)	F1 (%)
Regular Matching	Regular Matching	61.44	31.74	41.86
Semantic Similarity	Regular Matching	70.94	54.30	61.51
Sequence Labelling	Regular Matching	78.04	63.33	69.92
Regular Matching	Sentiment Analysis	68.81	37.12	48.22
FOSS-LTE	FOSS-LTE	51.34	60.08	55.37
LiDetector^{-}	LiDetector	80.23	66.75	72.87
LiDetector (Sequence Labelling)	**LiDetector (Sentiment Analysis)**	**85.88**	**69.70**	**76.95**

R.: Right; O.: Obligation; P: Precision; R: Recall; F1: F1 score.

license texts are written in natural language, methods of sentiment analysis cannot be directly applied to infer the stated conditions of software use.

By analyzing the results of FOSS-LTE, we found that FOSS-LTE does not distinguish between license terms and their attitudes. Instead, it predefines a set of phrases that represent regulations (e.g., *MustOfferSourceCode*), and utilizes a topic model to extract regulations from license texts. Compared with FOSS-LTE, LiDetector learns and extracts information from license texts with finer granularity. Specifically, it learns to identify license terms from texts, based on which it infers the implied attitudes towards the identified terms. By this means, LiDetector is capable of inferring regulations with more flexibility compared with predefined regulations.

For regular matching and LiDetector, we define the same set of attitude keywords, as well as the same order to search for the attitude keywords. However, we observe that simple regular matching is less effective than LiDetector in predicting attitudes. The characteristics of license sentences may contribute to the gap between performances of regular matching and LiDetector. Specifically, license sentences are typically long and complicated, which may contain massive tokens that are irrelevant to the attitudes towards a target term. Regular matching equally compares each token in the sentence with predefined keywords, which may introduce noise. In contrast, LiDetector filters out irrelevant tokens by grammar parsing, narrows down the scope of searching for permissive and restrictive expressions, and only analyzes powerful tokens that may have an influence on the attitude toward a target entity.

The overall performance for license comprehension. To further investigate the performance of LiDetector combining the first two phases, we also report the results over the entire process of license term identification and attitude inference. Specifically, given a license, we first identify license terms listed in Table 1. Then, the identified term entities, as well as the original license sentences, were fed into the second phase, so as to infer the attitudes towards these terms. We conduct the comparative study on the ground truth described in Section 4.1, and use precision, recall, and F1 score to evaluate the performances of the compared methods. Table 7 summarizes the results of each method. It can be seen that the combination of the methods used in LiDetector (i.e., sequence labelling and sentiment analysis) achieves the best performance among all combination methods, with 85.88% precision, 69.70% recall, and 76.95% F1 score.

Table 8. Results for Incompatibility Detection ("Conflict": two incompatible attitudes towards the same term.)

Method	#Pro.	#Identified pro.	#Conflicts	FP	FN
Ninka [15]	200	144 (117 overlap)	13,978 (12,509 overlap)	23 (15.97%)	35 (22.44%)
SPDX-VT [27]	200	71 (71 overlap)	–	6 (8.45%)	91 (58.33%)
Librariesio [33]	200	169 (137 overlap)	–	40 (23.67%)	27 (17.31%)
LiDetector (SPDX)	**200**	**157**	**14,586**	–	–
LiDetector (All)	**200**	**169**	**22,941**	**17 (10.06%)**	**4 (2.56%)**

***LiDetector* (SPDX)**: *the result for licenses only on SPDX;* ***LiDetector* (All)**: *the result for all the extracted licenses.*

4.4 Evaluation on Incompatibility Detection

4.4.1 ***Setup.*** To investigate the overall effectiveness of LiDetector in incompatibility detection, we also compare LiDetector with three state-of-the-art tools, i.e., the SPDX Violation Tools (**SPDX-VT**) [25, 27], **Ninka** [15] equipped with *tldrlegal* [30], and **Librariesio** [33], a license compatibility checking tool for SPDX licenses. Since the other three tools are all based on the licenses on SPDX due to their inability on custom licenses, to make a fair comparison, we conduct this experiment and show the result in two ways: (1) Only the official licenses listed in the Software Package Data Exchange (SPDX) [12] are considered for incompatibility detection. (2) All the licenses extracted from the project are considered (including the custom ones) for incompatibility detection.

Specifically, we randomly selected 200 projects from the 1,846 GitHub projects described in Section 2.3, and constructed a ground-truth dataset manually verified and cross-validated by three authors and a lawyer. In total, we extracted 1,298 unique licenses (including 191 project licenses and 1,107 component licenses), with 1,041 official licenses and 257 custom ones. The ground-truth dataset of incompatible projects has been made available online [66]. For each project, we first extract licenses in three forms (i.e., declared, referenced, and inline), and then use the aforementioned tools to detect the incompatibility issues. Note that, since we focus on license incompatibility detection for software projects, the FP and FN metrics are computed at the project level. Moreover, we also calculated the number of *conflicts* and *overlaps* for the these tools. In this paper, a *conflict* denotes a specific incompatibility issue, which means two incompatible attitudes towards the same license term. For instance, *cannot disclose source* versus *must disclose source.* We use *overlap* to denote the number of incompatible projects/conflicts that can also be detected by LiDetector.

4.4.2 ***Results.*** Table 8 shows the incompatibility detection results on SPDX licenses only and all the extracted licenses. As for the results on SPDX licenses only, it can be seen that LiDetector detects 14,586 conflicts in 157 projects, while SPDX-VT only finds license incompatibility in 71 out of 200 projects, all of which can be detected by LiDetector. Note that LiDetector filters all official licenses whose regulations have already been known. For this reason, when only considering the official licenses in SPDX [60], there are no false positives and false negatives. Since SPDX-VT only predefined a graph to denote the compliance relationships between a set of licenses, the output of SPDX-VT is relatively coarse-grained (i.e., without detailed explanation of how licenses conflict with each other). Therefore, the number of conflicts detected by SPDX-VT are not presented in Table 8. It can be seen that the number of false positives reported by SPDX-VT is 6. However, the FN rate of SPDX-VT is 58.33%, nearly 30 times higher than that of LiDetector for all extracted licenses. By analyzing the process of SPDX-VT, we found that it designs a strict rule that only a set of license texts are identified and analyzed to detect incompatibility. However, the graph defined by SPDX-VT only includes 20 popular licenses, other licenses can not be detected by SPDX-VT.

For the results on all extracted licenses, among 200 projects, LiDetector identified 22,941 conflicts in 169 projects, with 10.06% FP rate and 2.56% FN rate. Ninka equipped with tldrlegal identified 13,978 conflicts in 144 projects, with 15.97% FP rate and 22.44% FN rate. It can be concluded that LiDetector has the superiority over Ninka (equipped with *tldrlegal*) from the respects of both

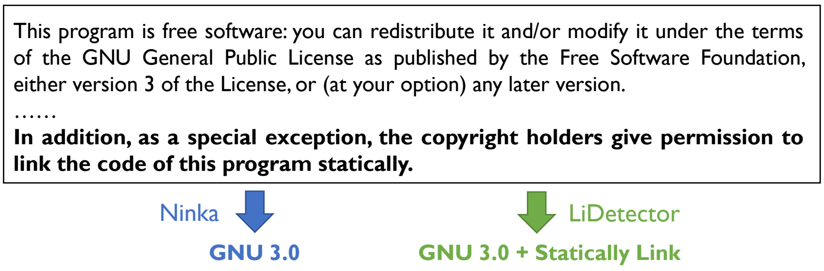

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
……
In addition, as a special exception, the copyright holders give permission to link the code of this program statically.

Fig. 9. A License Exception in BDF [4] Misidentified by Ninka

precision and recall. By analyzing the reported results of Ninka, we found that some licenses are customized by authors of software products, which cannot be identified by Ninka. For instance, the component license in Fig. 2a is a custom license that cannot be identified by Ninka. Therefore, Ninka equipped with *tldrlegal* cannot detect the incompatibility issue in Fig. 2a. In addition, some licenses are *exceptions* of common open source licenses (i.e., license variants generated by developers based on popular licenses, thus resulting in the modification of its term attitudes), which regulates different rights and obligations but misidentified by Ninka. Fig. 9 shows a custom license which is an exception of GNU 3.0. In this case, LiDetector first identified that the license referenced GNU 3.0; then, it fed the rest text into the probabilistic model and inferred "CAN Statistically Link" from the text. However, Ninka only identified the license as GNU 3.0 ignoring the custom exceptions. As a result, the accuracy of Ninka limits its effectiveness in detecting license incompatibility.

Librariesio [33] identified 169 incompatible projects, with 23.67% FP rate and 17.31% FN rate. Similar to SPDX-VT, the number of conflicts detected by Librariesio are also not presented in Table 8, since Librariesio provides no detailed explanation about how licenses conflict with each other. By analyzing the results of Librariesio and LiDetector, we found that the FP rate of Librariesio is more than twice as high than that of LiDetector and the FN rate of Librariesio is almost 8 times higher. The reason are two folds. First, Librariesio detects incompatibility between a set of SPDX licenses, while other licenses are ignored. Second, Librariesio detects incompatibility with a predefined set of heuristic rules, which might not be suited for large amounts of licenses which are flexible in expressions.

It can also be seen that the FP rate of LiDetector is 10.06%. By analyzing the false positives, we found that some license terms were not correctly identified from license texts, especially for license terms such as *Statically Link* and *Relicense*. The reason behind is that these license terms do not as frequently occur as other terms in the training dataset, which poses challenges in identifying these terms. We note that false negatives in the first phrase (i.e., license term identification) could lead to both false positives and false negatives in incompatibility detection. Another reason for false positives is the limitation of Stanford CoreNLP [20]. For instance, in the project Blosc [5], there is a license sentence:
"... In any action to enforce the terms of this License or seeking damages relating thereto, its costs and expenses, including, without limitation, reasonable attorneys' fees and costs incurred in connection with such action, including any appeal of such action, shall be recovered for the prevailing party ..."
From this sentence, LiDetector identified the license term *Compensate for Damages*. However, when inferring the attitudes towards this license term, Stanford CoreNLP [20] failed to parse the whole sentence, since the sentence is too long. The part of sentence *"shall be recovered for the prevailing party"* was ignored by the tool, leading to the inference results to be CAN instead of MUST. Due to the wrong parsing result, LiDetector failed to extract the obligation "MUST Compensate for Damages" from the project license, leading to a false positive.

Table 9. Overall Status of License Incompatibility

# Total projects	# Incompatible projects		# Conflicts	
1,846	1,346		75,249	
	PL vs CL	CL vs CL	PL vs CL	CL vs CL
	1,087 (80.76%)	259 (19.24%)	66,447 (88.30%)	8,802 (11.70%)

PL: Project License; CL: Component License

Table 10. The Number of Official and Custom Licenses Involved in Conflicts

	Official vs. Official	Custom vs. Official	Custom vs. Custom
# Conflicts	43,780	29,835	1,634
# Projects	1,324	384	13

5 EMPIRICAL STUDY ON COMPATIBILITY ANALYSIS

By leveraging LiDetector, we further conduct an empirical study on the 1,846 projects collected from Github, so as to investigate incompatibility issues in real-world OSS licenses.

5.1 Overall status of incompatibility issues

Among 1,846 projects, LiDetector detected 75,249 conflicts in 1,346 projects, i.e., **72.91%** projects suffer from license incompatibility issues according to LiDetector. The evaluation results can be seen in Table 9 and Table 10. Compared with the incompatible project rate (i.e., 48.86%) in the motivating study (Section 2.3), LiDetector is capable of detecting more incompatibility issues in OSS licenses owing to its ability for custom licenses.

To better understand the incompatibility status, we take an in-depth analysis on the incompatibility issues detected by LiDetector from two aspects: (1) incompatibility involving project licenses and component licenses (Table 9); and (2) incompatibility involving official licenses and custom licenses (Table 10). For the first aspect, we divide the incompatibility issues into two categories: a) incompatibility between component licenses, and b) incompatibility between a project license and a component license. We can see that 1,087 out of 1,846 projects were found to have compliance issues between their project licenses and their component licenses, while only 259 projects contain incompatibility issues between component licenses. We also show the exact number of conflicts detected by LiDetector in Table 9. The values in the column *PL vs CL* are the numbers of incompatibility issues between a project license and a component license. It can be seen that among all investigated projects, conflicts often occur between a project license and a component license. In other words, given an open source project, it is often the case where users who conform to the license of the whole project may violate licenses of some components. The difficulty to combine licenses from different components into the whole project license may account for this phenomenon. The investigation results encourage developers to pay more attention to creating the license for the whole project especially when they integrate third-party software components with licenses in their projects.

Table 10 shows the detailed result about the number of official licenses and custom licenses involved in license incompatibility. It can be seen that 1,324 projects have incompatibility issues between a pair of official licenses; 384 projects have incompatibility issues between an official license and a custom license; only 13 projects contains conflicts between a pair of custom licenses. The results are consistent with the observation that a majority of licenses are based on official licenses. We can also observe that custom licenses are involved in 31,469 conflicts and 397 projects. In other words, even all official licenses are compatible with each other, there are still near 21.5% of 1,846 projects that have incompatibility issues.

Impact of the default attitude towards an absence term. As aforementioned in Section 3.5, following the rule declared by *choosealicense* [8], if a license term does not appear in the license

Table 11. Incompatible Projects with Different Sizes.

Pro. Size (KB)	#Pro.	#Incomp. pro.	#Conflicts
<= 999	840	550 (65.48%)	12,899
1,000~4,999	551	426 (77.31%)	20,757
5,000~9,999	281	219 (77.94%)	12,081
>= 10,000	174	151 (86.78%)	29,512
Total	1,846	1,346	75,249

Table 12. Projects with Different Numbers of CLs.

#CL	#Pro.	#Incomp. pro.	#Conflicts
<= 24	887	479 (54.00 %)	7,283
25~49	495	428 (86.46%)	11,272
50~99	316	294 (93.04%)	12,503
>= 100	148	145 (97.97%)	44,191
Total	1,846	1,346	75,249

according to LiDetector, the default attitude towards this term is set to CANNOT, which implies that nobody can copy, distribute, or modify the work. To further investigate the impact of such a default setting, we also conduct an ablation study where the default attitude of absent terms is set to CAN. The results show that when the default attitude of absent terms is CAN, LiDetector detected 1,104 incompatible projects with 10,507 license incompatibility issues. When the default setting is CANNOT, LiDetector detected 1,346 incompatible projects with 75,249 incompatibility issues. It can be seen that when the default attitude is set to CANNOT, LiDetector detected more incompatible projects with seven times of the number of incompatibility issues. A possible reason is that there exist more restrictive statements than permissive statements when the default setting of absent terms is set to CANNOT. Nevertheless, it has more influence on the number of detected incompatibility issues than incompatible projects. A large number of absent license terms in component licenses might account for such difference.

The impacts of project size and the number of component licenses. To analyze the distribution of incompatibility issues from other dimensions, e.g., project size and the number of component licenses, we divide projects into different groups and show the results in Table 11 and Table 12. From Table 11, we can see that projects that have larger size are more likely to trigger license incompatibility, which is because larger projects possibly own more imported packages and more complex dependency construction [36]. From Table 12, we can see that projects that have more component licenses are more likely to trigger license incompatibility. Especially for projects containing more than 100 component licenses, almost all of these projects (i.e., 97.97%) have license incompatibility issues. The results show that projects with many components need to be noticed for the potential risk towards license incompatibility.

5.2 Top licenses with incompatibility issues

To analyze the relationships between incompatibility and license types, we counted the number of licenses involved in the incompatibility issues. Specifically, we found that common open source licenses (*official licences*) are incompatible with other licenses in 1,346 projects with 117,395 conflicts, and custom licenses are incompatible with other licenses in 384 projects with 33,103 conflicts. Note that we used Ninka [15] to identify well-known licenses. Each conflict occurs between two licenses, and the total number of conflicts detected by LiDetector is still 75,249 The reason why well-known licenses contribute to more than half of incompatibility issues is twofold: first, well-known licenses are frequently used in OSS; second, third-party packages and libraries incorporated by developers are often accompanied with common licenses, which may induces incompatibility with the project license. Nevertheless, custom licenses and exceptions also account for a considerable number of incompatibility issues, which indicates that relying on well-known licenses to detect incompatibility is not effective enough. A tool that is capable of analyzing compatibility for both well-known and custom licenses are needed for practical use. Finally, we counted the number of each license to be involved in the incompatibility issues, and sorted them from high to low. We found that the top 5 common open source licenses are *MIT License* [57], *Zope Public License 2.1* [58], *Apache License 2.0* [53], *GNU Lesser General Public License v3* [56], and *BSD 3-Clause License* [54]. For example, in the project HaboMalHunter [23] , the project license is the MIT License that states

Table 13. Top 10 Terms with Incompatibility Issues

ID	Term	#Conflicts	#Pro.	Incompatibility Type
1	Sublicense	7,801	604	Can↔Cannot
2	Use Trademark	7761	933	Can↔Cannot, Can↔Must, Cannot↔Must
3	Commercial Use	7,418	720	Can↔Cannot
4	Distribute	6,716	824	Can↔Cannot
5	Modify	6,564	749	Can↔Cannot
6	Place Warranty	5,544	489	Can↔Cannot
7	Include Copyright	4,741	1,303	Cannot↔Must
8	Include License	4,312	1,306	Cannot↔Must
9	State Changes	2,714	969	Can↔Must, Cannot↔Must
10	Disclose Source	2,670	1,085	Can↔Cannot, Can↔Must, Cannot↔Must

"CAN Sublicense", while one of the component licenses is the Wizardry License that regulates "CANNOT Sublicense". In this case, developers who conform to the project license may still violate the component license, which leads to incompatibility issues.

5.3 Top 10 license terms with incompatibility issues

To further analyze the reasons of the incompatibility issues, we list the top 10 license terms involved in license incompatibility in Table 13. For each license term, we show the number of incompatible projects (#Pro.), the number of conflicts (#Conflicts), and the type of incompatibility (Incompatibility Type). It can be seen that *Sublicense*, *Use Trademark*, and *Commercial Use* are the top 3 license terms that lead to incompatibility issues, affecting 7,801, 7,761, and 7,418 conflicts, respectively. For *Sublicense*, we find that the incompatibility typically occurs when a project license indicates that users CAN incorporate the work into works with more restrictive licenses, while its component license declares CANNOT. The same can be seen in *Commercial Use*, where a project license allows using the project for commercial purposes, while its component licenses declares CANNOT. Developers need to pay attention to these license terms especially when they incorporate third-party software packages accompanied with licenses, since unauthorized use of such packages may lead to the legal and financial risks.

We can also observe that *Include License* is involved in projects most frequently (i.e., in 1,306 projects with license incompatibility), which is very common license term among most open source software licenses. For *Include License*, the incompatibility usually occurs between the two component licenses; one license component declares that users MUST include the full text of license in modified software, while another component license declares CANNOT.

We also investigated the top 5 license terms with incompatibility issues for official licenses and custom licenses, respectively. As shown in Table 14, there are some differences between the top license terms with conflicts of official licenses and custom licenses. Specifically, the top 3 terms of custom licenses are *Distribute*, *Commercial Use*, and *Modify*, while the top 3 terms of official licenses are *Use Trademark*, *Sublicense*, and *Commercial Use*. We can also observe that in both official and custom licenses, top 5 conflict terms are all right-related terms. The results encourage developers to pay more attention on the rights conveyed by licenses. Finally, some license terms are not frequently involved in conflicts. It does not mean that these license terms are more reliable than others, since the reason behind might be that they are rarely used in licenses.

Table 14. Top 5 Terms with Incompatibility Issues in Official Licenses and Custom Licenses

No.	1	2	3	4	5
Official	Use Trademark	Sublicense	Commercial Use	Modify	Distribute
Custom	Distribute	Commercial Use	Modify	Sublicense	Use Trademark

6 DISCUSSION

In this section, we first discuss the lessons learned from perspectives of different stakeholders (i.e., developers, commercial product managers), and then discuss the limitations and threats to validity.

6.1 Lessons Learned

From the perspective of developers, although open source software facilitates software development, developers need to pay more attention to the compatibility between multiple licenses, especially when incorporating third-party software packages. **First**, as revealed in the empirical study (Section 5), over 70 percent of popular projects in GitHub suffer from the problem of license incompatibility, each of which contains 56 pairs of conflict licenses in average. **Second**, developers need to carefully select the project license, since more than 80% conflicts occur between the project license and other licenses. Especially for projects incorporating third-party software packages, developers are expected to check all component licenses and select the license that are more restrictive than component licenses as the project license. For instance, although widely-used and very permissive, licenses such as the MIT License are often involved in incompatibility issues when they are used as a project license. **Third**, licenses may have different versions and exceptions that regulate different rights and obligations. Developers are also allowed to create their own licenses based on some official licenses (i.e., exceptions). Relying on license identification tools such as Ninka [15] may lead to misunderstandings of license texts, resulting in license incompatibility issues. LiDetector can help developers understand the semantics of license texts automatically and then detect license incompatibility within a project. With the assistant of LiDetector, developers can avoid infringements of laws and rules when reusing existing knowledge.

From the perspective of commercial product manager, although common open source licenses are popular in the OSS community, authors can also create their own licenses, which are difficult to be identified with existing tools. When reusing knowledge from OSS, companies should be aware of the risk of incompatible licenses, especially the attitudes of authors towards *Sublicense*, *Use Trademark*, and *Commercial Use* as revealed in Section 5. With the ability to detect license incompatibility, LiDetector can help commercial companies examine the compatibility between multiple licenses accurately, so as to avoid financial and legal risks. In addition, with LiDetector, product managers can acquire license terms and attitudes implied by each license, which provide useful information for product managers to select an appropriate license or create a new license for their software products.

6.2 Limitations and Threats to Validity

Limitations. LiDetector utilizes the *Stanford CoreNLP* tool [20] to preprocess and parse license sentences. For this reason, the effectiveness of *Stanford CoreNLP* affects the accuracy of LiDetector. For instance, by analyzing the evaluation results, we found that when the target sentence is too long and complicated, *Stanford CoreNLP* only parsed parts of the sentence, which affects the effectiveness of LiDetector. In addition, we construct a probabilistic model to identify license terms, therefore it cannot be ensured that all license terms can be identified by the proposed model. Nevertheless, from our experimental results, we can see LiDetector failed to detect incompatibility issues in only 17 cases, with 10.06% FP rate and 2.56% FN rate. Therefore, LiDetector is still effective to detect license incompatibility for most projects. Another limitation is that LiDetector cannot filter out irrelevant licenses. For example, for files that are only for testing purposes and are not really component files, if they contain licenses and should not be considered for incompatibility detection, LiDetector cannot filter them out. Moreover, for licenses that are neither designated in the target projects nor obtained by LiDetector from external resources, LiDetector fails to collect these licenses and detect incompatibility. For instance, although LiDetector collects licenses accompanied with imported packages, for packages whose sources cannot be found, LiDetector can not collect them and detect incompatibility. Another example is that if derivative works only reuse OSS code without any licenses or references, LiDetector cannot obtain licenses and thus cannot detect license incompatibility in this case. Some related research directions such as clone detection [32] and license compliance detection [14] address such issues, while this paper only focuses on incompatibility between multiple licenses within the same project. Finally, for referenced licenses, we collect licenses accompanied with imported packages, while ignoring packages which are further imported by these packages.

Threats to Validity. (1) LiDetector identifies 23 types of license terms from each license to detect incompatibility. Nevertheless, there could be a few cases where authors of software products have special requirements outside the scope of LiDetector. (2) In addition, LiDetector may detect some incompatibility issues based on the inaccurate results from the previous two phases (i.e., term identification, and right and obligation inferring), leading to false positives. However, according to our experiments, the accuracy of the two phases achieves over 90%, and the incompatibility detection accuracy also reaches over 90%, which is much higher than state-of-the-art tools. (3) Another threat may be that due to the substantial manual effort of labelling, we were not able to collect a large-scale dataset for license entity tagging. However, we employ a semi-supervised learning method that incorporates unlabelled samples for training. The experimental results show that LiDetector achieves 93.28% precision and 83.58% F1 score when identifying license terms, which outperforms the baselines in this phase. (4) The performance of LiDetector was evaluated on a ground-truth dataset comprising 200 popular projects, and the quality of the test set may threaten the results. To mitigate this problem, three experts manually verified and cross-validated the dataset, which has been made available online for further study [66].

7 RELATED WORK

7.1 Semantic Extraction

License texts are typically long and complicated, which are not straightforward for developers to understand. To facilitate the process of understanding and choosing licenses, much research has been done to extract license semantics. The previous studies can be categorized into two groups.

The ontology study. Alspaugh et al. [1] [2] extracted tuples (e.g., actor, action, and object) from license texts to model 10 licenses. Gordon et al. [9] used the Web Ontology Language (OWL) tool to build the ontology for OSS licenses and projects. With manual analysis, the constructed ontology contains knowledge from 8 popular licenses. Based on this ontology, the authors further analyzed license compatibility for open source software [17], and developed the MARKOS license analyzer [18] [19]. These studies typically extracted information from license texts by manually analyzing several licenses, which limits its application scope.

Term extraction. FindOSSLicense [25] classified and summarized license sentences in 24 license texts through manual analysis, and obtained terms to model licenses. Based on a topic model, FOSS-LTE [28] identified license terms by mapping licenses terms with sentences, and mapping sentences with topics by Latent Dirichlet Allocation (LDA). By this means, it built the relationships between terms and topics. FOSS-LTE is most related to this work, focusing on the automated extraction of license terms to help developers better understand of licenses. However, the simple topic model employed by FOSS-LTE may introduce much noise, which could be a possible explanation for the low accuracy of FOSS-LTE. In contrast, LiDetector prepossesses licenses with a clustering technique, followed by a two-phase learning method (i.e., term entity extraction, right and obligation inference), which is capable of extracting licenses flexible in expressions. In addition, LiDetector employs semi-supervised learning to train the term entity extraction model so as to save the labelling effort and enhance performance.

7.2 License Identification

In order to facilitate the understanding of licenses, some studies attempted to identify common licenses automatically. Gobeille et al. [16] implemented FOSSology that uses a binary Symbol Alignment Matrix algorithm (bSAM) to identify licenses. Tuunanen et al. [62] developed a tool called ASLA to identify software licenses based on regular expressions, and analyze the interactions between a set of licenses. Similarly, Xu et al. [65] proposed regular matching to identify licenses including their names and versions. German et al. [15] developed Ninka for automated license identification based on sentence matching. Higashi et al. [24] exploited the cluster learning method to further identify licenses marked as *unknown* by Ninka, as a supplementary study.

Despite the progress, previous studies on license identification requires much prior knowledge from experts, and can only be applied on a predefined set of licenses. In contrast, LiDetector learns to extract the semantics of licenses from a large corpus of license sentences, which equips it with the capability to analyze arbitrary licenses which are flexible in expressions.

7.3 License Incompatibility Detection

To avoid legal and financial risks, recently, much research has been done to detect license incompatibility. A major of studies on license incompatibility detection are graph-based approaches [26, 27, 41, 63]. Typically, these studies manually constructed a graph to describe the compatibility relationships between a set of licenses, and detected incompatibility between these licenses by traversing the nodes and edges of the graph. For instance, Paschalides and Kapitsaki [41] proposed a tool named SLVT, which is based on the directed acyclic license graph, to examine license violations that may exist in a single or multiple SPDX files. Although strict and efficient, there exist a large number of licenses in real-word OSS, including various versions, exceptions, and custom licenses. Therefore, it is difficult to manually analyze the relationships between all licenses. Unlike graph-based approaches, LiDetector is a flexible and extensible tool that can be applied on an arbitrary license without prior knowledge, which enlarges its application scope.

8 CONCLUSION

In this paper, we propose LiDetector, an automated and effective tool to detect license incompatibility for open source software. It first identifies license terms and then infers the attitudes of authors towards identified terms. The experimental results demonstrate the effectiveness of LiDetector in incompatibility detection for OSS. We also study the license incompatibility status on 1,846 real-world open source projects by leveraging LiDetector, and analyze the characteristics of incompatibility issues and licenses. Our large-scale empirical study on 1,846 projects reveals that 72.91% of the projects are suffering from license incompatibility, including popular ones such as the MIT License and the Apache License. We highlighted lessons learned from perspectives of different stakeholders. All the datasets [66] and the replication package [67] are released for follow-up research. We believe LiDetector can benefit different stakeholders (e.g., developers) in terms of license compatibility.

ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation of China (No. 62102197) and National Key Research Project of China (No. 2021YFF0307202 and No. 2020YFB1005700).

REFERENCES

[1] Thomas A Alspaugh, Hazeline U Asuncion, and Walt Scacchi. 2009. Intellectual property rights requirements for heterogeneously-licensed systems. In *Proceedings of the 17th IEEE International Requirements Engineering Conference*. 24–33.

[2] Thomas A Alspaugh, Walt Scacchi, and Hazeline U Asuncion. 2010. Software licenses in context: The challenge of heterogeneously-licensed systems. *Journal of the Association for Information Systems* 11, 11 (2010), 2.

[3] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. 2019. Policylint: investigating internal privacy policy contradictions on Google Play. In *Proceedings of the 28th USENIX Conference on Security Symposium*. 585–602.

[4] BDF. 2021. The Backdoor Factory. https://github.com/secretsquirrel/the-backdoor-factory.

[5] Blosc. 2021. A blocking, shuffling and lossless compression library. https://github.com/Blosc/c-blosc.

[6] Sen Chen, Lingling Fan, Guozhu Meng, Ting Su, Minhui Xue, Yinxing Xue, Yang Liu, and Lihua Xu. 2020. An empirical assessment of security risks of global android banking apps. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1310–1322.

[7] Sen Chen, Ting Su, Lingling Fan, Guozhu Meng, Minhui Xue, Yang Liu, and Lihua Xu. 2018. Are mobile banking apps secure? what can be improved?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 797–802.

[8] choosealicense. 2012. Choose an open source license. https://choosealicense.com/no-permission/.

[9] Gordon Thomas F. 2010. Report on prototype decision support system for oss license compatibility issues. *Qualipso* (2010), 80.

[10] facebookarchive. 2021. Augmented Traffic Control. https://github.com/facebookarchive/augmented-traffic-control.

[11] Runyu Fan, Lizhe Wang, Jining Yan, Weijing Song, Yingqian Zhu, and Xiaodao Chen. 2020. Deep learning-based named entity recognition and knowledge graph construction for geological hazards. *ISPRS International Journal of Geo-Information* (2020).

[12] Linux Foundation. 2018. The Software Package Data Exchange. https://spdx.dev/.

[13] GR Gangadharan, Vincenzo D'Andrea, Stefano De Paoli, and Michael Weiss. 2012. Managing license compliance in free and open source software development. *Information Systems Frontiers* (2012), 143–154.

[14] Daniel German and Massimiliano Di Penta. 2012. A method for open source license compliance of java applications. *IEEE software* 29, 3 (2012), 58–63.

[15] Daniel M. German, Yuki Manabe, and Katsuro Inoue. 2010. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. 437–446.

[16] Robert Gobeille. 2008. The fossology project. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*. 47–50.

[17] Thomas F. Gordon. 2011. Analyzing open Source license compatibility issues with Carneades. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law*. 51–55.

[18] Thomas F. Gordon. 2013. Introducing the Carneades web application. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law*. 243–244.

[19] Thomas F. Gordon. 2014. A demonstration of the MARKOS license analyser. In *Proceedings of the 5th International Conference on Computational Models of Argument*. 461–462.

[20] Stanford NLP Group. 2020. corenlp. https://stanfordnlp.github.io/CoreNLP/.

[21] Hao Guo, Sen Chen, Zhenchang Xing, Xiaohong Li, Yude Bai, and Jiamou Sun. 2021. Detecting and Augmenting Missing Key Aspects in Vulnerability Descriptions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2021).

[22] Hao Guo, Zhenchang Xing, Sen Chen, Xiaohong Li, Yude Bai, and Hu Zhang. 2021. Key aspects augmentation of vulnerability description based on multiple security databases. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 1020–1025.

[23] HaboMalHunter. 2021. Habo Linux Malware Analysis System. https://github.com/Tencent/HaboMalHunter.

[24] Yunosuke Higashi, Yuki Manabe, and Masao Ohira. 2016. Clustering OSS license statements toward automatic generation of license rules. In *Proceddings of the 7th International Workshop on Empirical Software Engineering in Practice*. 30–35.

[25] Georgia Kapitsaki and Georgia Charalambous. 2019. Modeling and recommending open source licenses with find-OSSLicense. *IEEE Transactions on Software Engineering* (2019).

[26] Georgia M. Kapitsaki and Frederik Kramer. 2014. Open source license violation check for SPDX files. In *Software Reuse for Dynamic Systems in the Cloud and Beyond*. 90–105.

[27] Georgia M. Kapitsaki, Frederik Kramer, and Nikolaos D. Tselikas. 2017. Automating the license compatibility process in open source software with SPDX. *Journal of Systems and Software* (2017), 386 – 401.

[28] Georgia M. Kapitsaki and Demetris Paschalides. 2017. Identifying terms in open source software license texts. In *Proceedigns of the 24th Asia-Pacific Software Engineering Conference*. 540–545.

[29] Petros Karvelis, Dimitris Gavrilis, George Georgoulas, and Chrysostomos Stylios. 2018. Topic recommendation using Doc2Vec. In *2018 International Joint Conference on Neural Networks*. 1–6.

[30] kevin. 2012. Software Licenses in Plain English. https://tldrlegal.com/.

[31] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*. 1188–1196.

[32] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. 2017. Cclearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 249–260.

[33] librariesio. 2015. Check compatibility between different SPDX licenses for checking dependency license compatibility. https://github.com/librariesio/license-compatibility.

[34] Open Source Licensing. 2004. *Software freedom and intellectual property law.*

[35] Ling. 2003. Alphabetical list of part-of-speech tags used in the Penn Treebank Project. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[36] Chengwei Liu, Sen Chen, Lingling Fan, Bihuan Chen, Yang Liu, and Xin Peng. 2022. Demystifying the Vulnerability Propagation and Its Evolution via Dependency Trees in the NPM Ecosystem. In *2022 IEEE/ACM 44nd International Conference on Software Engineering (ICSE)*. IEEE.

[37] Fabio Mancinelli, Jaap Boender, Roberto Di Cosmo, Jerome Vouillon, Berke Durak, Xavier Leroy, and Ralf Treinen. 2006. Managing the complexity of large free and open source package-based software distributions. In *21st IEEE/ACM International Conference on Automated Software Engineering*. 199–208.

[38] Arunesh Mathur, Harshal Choudhary, Priyank Vashist, William Thies, and Santhi Thilagam. 2012. An empirical study of license violations in open source projects. In *Proceedings of the 35th Annual IEEE Software Engineering Workshop*. 168–176.

[39] nltk. 2021. Natural Language Toolkit. https://www.nltk.org/.

[40] Opensource. 2021. What is open source? https://opensource.com/resources/what-open-source.

[41] Demetris Paschalides and Georgia M Kapitsaki. 2016. Validate your SPDX files for open source license violations. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 1047–1051.

[42] paul. 2021. Full extractor of class/interface/method definitions. https://github.com/paul-hammant/qdox.

[43] pivotal. 2021. Find licenses for your project's dependencies. https://github.com/pivotal/LicenseFinder.

[44] ProgrammerSought. 2021. *The first case of GPL agreement in China is settled. How should the relevant open source software be controlled?*

[45] PyPi. 2021. Find, install and publish Python packages with the Python Package Index. https://pypi.org/.

[46] Jaideep Reddy. 2015. The Consequences of Violating Open Source Licenses. https://btlj.org/2015/11/consequences-violating-open-source-licenses/.

[47] Nils Reimers and Iryna Gurevych. 2017. Optimal hyperparameters for deep LSTM-networks for sequence labeling tasks. *arXiv e-prints* (2017). arXiv:1707.06799

[48] Christopher D. Manning Richard Socher. 2014. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/.

[49] robinhood. 2021. Faust. https://github.com/robinhood/faust.

[50] Colin Scicluna, James de la Higuera. 2016. Grammatical inference of PCFGs applied to language modelling and unsupervised parsing. *Fundamenta Informaticae* (2016), 379–402.

[51] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 1631–1642.

[52] Georgios S. Solakidis, Konstantinos N. Vavliakis, and Pericles A. Mitkas. 2014. Multilingual sentiment analysis using emoticons and keywords. In *Proceedings of the IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*. 102–109.

[53] SPDX. 2018. Apache License 2.0. https://spdx.org/licenses/Apache-2.0.html.

[54] SPDX. 2018. BSD 3-Clause "New" or "Revised" License. https://spdx.org/licenses/BSD-3-Clause.html.

[55] SPDX. 2018. Creative Commons Attribution Share Alike 4.0 International. https://spdx.org/licenses/CC-BY-SA-4.0.html.

[56] SPDX. 2018. GNU Lesser General Public License v3.0 only. https://spdx.org/licenses/LGPL-3.0-only.html.
[57] SPDX. 2018. The MIT License. https://spdx.org/licenses/MIT.html.
[58] SPDX. 2018. Zope Public License 2.1. https://spdx.org/licenses/ZPL-2.1.html.
[59] SPDX. 2021. Creative Commons Attribution 3.0 Unported. https://spdx.org/licenses/CC-BY-3.0.html.
[60] SPDX. 2021. SPDX License List. https://spdx.org/licenses/.
[61] Statsite. 2021. Statsite. https://github.com/statsite/statsite.
[62] Tuunanen Timo, Koskinen Jussi, and Kärkkäinen Tommi. 2009. Automated software license analysis. *Automated Software Engineering* 16 (2009), 455–490.
[63] David A. Wheeler. 2007. The free-libre / open source software (FLOSS) license slide. http://www.dwheeler.com/essays/floss-license-slide.pdf.
[64] Linzhong Xia, Jun Liu, and Zhenjiu Zhang. 2019. Automatic essay scoring model based on two-layer bi-directional long-short term memory network. In *Proceedings of the 2019 3rd International Conference on Computer Science and Artificial Intelligence*. 133–137.
[65] HongBo Xu, HuiHui Yang, Dan Wan, and JiangPing Wan. 2010. The design and implement of open source license tracking system. In *Proceddings of the 2010 International Conference on Computational Intelligence and Software Engineering*. 1–4.
[66] Sihan Xu, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu, and Hua Ji. 2021. LiDetector: License Incompatiblity Detection for Open Source Software. https://sites.google.com/view/lidetector.
[67] Sihan Xu, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu, and Hua Ji. 2021. LiDetector: License Incompatiblity Detection for Open Source Software. https://github.com/XuSihan/LiDetector.
[68] Xian Zhan, Lingling Fan, Sen Chen, Feng Wu, Tianming Liu, Xiapu Luo, and Yang Liu. 2021. Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1695–1707.
[69] Xian Zhan, Lingling Fan, Tianming Liu, Sen Chen, Li Li, Haoyu Wang, Yifei Xu, Xiapu Luo, and Yang Liu. 2020. Automated third-party library detection for android applications: Are we there yet?. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 919–930.
[70] Xian Zhan, Tianming Liu, Lingling Fan, Li Li, Sen Chen, Xiapu Luo, and Yang Liu. 2021. Research on Third-Party Libraries in Android Apps: A Taxonomy and Systematic Literature Review. *IEEE Transactions on Software Engineering* (2021).