A Large-Scale Empirical Study of Open Source License Usage: Practices and Challenges

Jiaqi Wu
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, Zhejiang, China
jiaqiwu@zju.edu.cn

Lingfeng Bao*
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, Zhejiang, China
lingfengbao@zju.edu.cn

Xiaohu Yang
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, Zhejiang, China
yangxh@zju.edu.cn

Xin Xia
Huawei
China
xin.xia@acm.org

Xing Hu
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, Zhejiang, China
xinghu@zju.edu.cn

ABSTRACT

The popularity of open source software (OSS) has led to a significant increase in the number of available licenses, each with their own set of terms and conditions. This proliferation of licenses has made it increasingly challenging for developers to select an appropriate license for their projects and to ensure that they are complying with the terms of those licenses. As a result, there is a need for empirical studies to identify current practices and challenges in license usage, both to help developers make informed decisions about license selection and to ensure that OSS is being used and distributed in a legal and ethical manner. Moreover, the development of new licenses might be required to better meet the needs of the open source community and address emerging legal issues.

In this paper, we conduct a large-scale empirical study of license usage across five package management platforms, i.e., Maven, NPM, PyPI, RubyGems, and Cargo. Our objective is to examine the current trends and potential issues in license usage of the OSS community. In total, we analyze the licenses of 33,710,877 packages across the selected five platforms. We statistically analyze licenses in package management platforms from multiple perspectives, e.g., license usage, license incompatibility, license updates, and license evolution. Moreover, we conduct a comparative study of various aspects of core packages and common packages in these platforms. Our results reveal irregularities in license names and license incompatibilities that require attention. We observe both similarities and differences in license usage across the five platforms, with Cargo being the most standardized among them. Finally, we discuss some implications for actions based on our findings.

KEYWORDS

OSS Licenses, Empirical Study, Package Management Platform

ACM Reference Format:
Jiaqi Wu, Lingfeng Bao, Xiaohu Yang, Xin Xia, and Xing Hu. 2024. A Large-Scale Empirical Study of Open Source License Usage: Practices and Challenges. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3643991.3644900

1 INTRODUCTION

The use of open source software (OSS) has become increasingly prevalent in recent years due to its many benefits, such as cost savings, flexibility, and the ability to collaborate with a large community of developers. OSS licenses that comply with the Open Source Definition [5] play an important role in ensuring the sustainability of the OSS community [23, 34] because they provide legal terms and conditions under which the software can be used, modified, and distributed by others, while also protecting the rights of the original creator.

However, the variety of OSS licenses is constantly expanding, with many organizations and individuals developing their own licenses to cater to specific needs and situations. There are currently over 450 different licenses in existence, according to the SPDX license list [38] which is maintained by the Linux Foundation's SPDX project. This comprehensive list comprises commonly found licenses and exceptions used in free and open or collaborative software, data, hardware, or documentation. Each license contains its own set of terms and conditions, posing a great deal of challenges and complexities for software developers and businesses.

Moreover, the reuse of OSS components or modules to construct a software system can be a cost-effective approach for businesses to minimize software development expenses [19, 32]. However, integrating third-party OSS components can easily trigger legal

**Corresponding authors

risks [47, 48]. Various OSS software and tools employ distinct licenses, failing to adhere to the respective obligations during the integration of third-party OSS components can result in license incompatibility issues [24].

Many previous studies have investigated license usage in OSS software, but most of them were not conducted in the data of the package management platform [22, 31, 34, 41]. Instead, compared with the data from other environments such as GitHub which are often personal projects and class projects [26], the data from the package management platform are often more mature. Meloca et al. [33] studied the use of non-OSI-approved licenses in package management platforms. They investigated the usage, impact, and adoption of non-OSI-approved licenses in three package management platforms (i.e., NPM [3], RubyGems [4], and CRAN [1]), and found that most of the non-OSI-approved licenses were related to the absence of a license. Their study only focused on non-OSI-approved licenses and not on more OSI licenses, while they categorized the absence of licenses as non-OSI-approved licenses without further differentiation studies. Qiu et al. [36] conducted an empirical study on NPM [6] data to investigate the prevalence of dependency-related license violations and conducted a preliminary questionnaire on the authors of packages detected as having dependency-related license violations. However, their study was conducted only on NPM, and the content and scope of the questionnaire were too small to be statistically significant.

In this paper, we conduct a large-scale empirical study to investigate the usage of OSS licenses by building a large dataset, which contains the metadata of all packages in five popular platforms, i.e., Maven [3], NPM [6], PyPI [2], RubyGems [4], and Cargo [7]. For each version of a package, we extract its metadata including the declared license information, release time, its dependencies, etc. The constructed dataset consists of a total of 33,710,877 versioned packages across the selected five platforms.

We design an algorithm to accurately extract the license information for each versioned package. Our algorithm is capable of distinguishing *SPDX licenses*, *unspecified licenses*, and *incomplete licenses*. A *SPDX license* means a license that clearly and unambiguously provides the full name or identifier of the license in the SPDX license list. *Unspecified licenses* refer to licenses that do not provide any licensing information or provide information that cannot be identified. *Incomplete licenses*, on the other hand, are licenses that do not provide the full name of the license but provide only a partial name without specifying the version, for example, "BSD" or "GNU GPL".

In this study, we statistically analyze the license usage of package management platforms from multiple perspectives. First, we analyze basic information about license usage across the selected platforms, such as the ratio of SPDX licenses and license type preferences. Our findings reveal a high percentage of unspecified and incomplete licenses, and each platform exhibits its own unique preferences in license usage, as well as some commonalities. Second, we investigate license changes and find that the license change ratio is not high. We also study license changes between single and multi-licenses as well as license changes with different restrictions. Third, we check incompatibility issues in packages across different platforms with two tools (i.e., LiDetector [50] and License-compatibility [29]). We find that the incompatibility rate is below 8% for all platforms and that the clause most likely to cause incompatibility is *Sublicense*, which pertains to the ability of developers to grant/extend a license to the software. Fourth, we analyze license evolution with the temporal information of packages to determine the life cycle and popularity of each license and the evolution of the most popular licenses over time. We find that the annual growth rate of distinct licenses has slowed and that Apache-2.0 is on track to overtake MIT as the most popular license. Finally, we identify the core packages for different platforms by building a dependency graph of the packages and analyze the differences in license usage between the overall packages and the core packages. We find more license incompatibility in core packages on platforms other than Cargo, which in general is the most standardized package management platform for licenses.

We make the following contributions in our study:

- We conduct a statistical analysis of the usage of licenses in the current open-source landscape, based on data from five package management platforms. We examine the preferences of different package management platforms for different licenses. We provide a replication package [14] to foster future work, in line with good research practices.
- We track the trends in the usage of different licenses over time, as well as the evolution in the usage of licenses over time across the five package management platforms.
- We examine the results of license incompatibility among the overall and popular versioned packages in the five package management platforms, and analyze the terms that are most likely to cause license incompatibility.

2 EXPERIMENT SETUP

2.1 Dataset

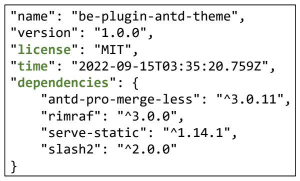
In this study, we select five package management platforms, i.e., Maven, NPM, PyPI, RubyGems, and Cargo. All these platforms are popular and provide convenient APIs or data dumps to collect their versioned package metadata. We collect the metadata of all the versioned packages for each platform as follows:

- **Maven**: We get a complete Maven package list from Maven's official website [9]. For each package in the list, we download the *pom.xml* file that contains its metadata.
- **NPM**: We get the complete NPM package list through the node.js mirror service [10], and then crawl the metadata of each package.
- **PyPI**: We collect all the metadata of packages from a dataset hosted on the Google Big Query [11].
- **RubyGems**: We get the metadata of all packages from the weekly dump of the RubyGems.org PostgreSQL data [12].
- **Cargo**: We get the package metadata from its daily database dump [8].

The metadata of Maven, NPM and PyPI is in JSON format, and the metadata of RubyGems and Cargo is in PostgreSQL database file format. Figure 1 shows two examples of metadata and the contents have been partially adjusted to show. The cut-off date for all platforms is unified on December 31, 2022. Based on the collected metadata, we extract the following fields for each versioned package, i.e., name, version, license, dependencies, and release time. However, the license information may not be included in the POM

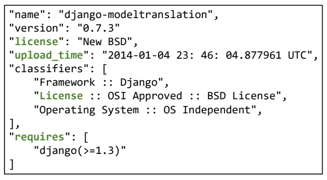
Table 1: Overall statistics of five platforms.

Platform	#Package	#Versioned Package
Maven	513,939	10,288,841
NPM	2,182,959	29,750,552
PyPI	485,223	4,413,827
RubyGems	188,807	1,427,793
Cargo	103,850	709,755



```
"name": "be-plugin-antd-theme",
"version": "1.0.0",
"license": "MIT",
"time": "2022-09-15T03:35:20.759Z",
"dependencies": {
    "antd-pro-merge-less": "^3.0.11",
    "rimraf": "^3.0.0",
    "serve-static": "^1.14.1",
    "slash2": "^2.0.0"
}
```

(a) NPM.



```
"name": "django-modeltranslation",
"version": "0.7.3"
"license": "New BSD",
"upload_time": "2014-01-04 23: 46: 04.877961 UTC",
"classifiers": [
    "Framework :: Django",
    "License :: OSI Approved :: BSD License",
    "Operating System :: OS Independent",
],
"requires": [
    "django(>=1.3)"
]
```

(b) PyPI.

Figure 1: Sample metadata for NPM and PyPI.

file in Maven, or it may be provided in the form of comments rather than in a specific field. We discovered that the official Maven repository [13] has marked licenses for most repositories. Unfortunately, there is currently no accessible API to fetch this information. Furthermore, the time details are absent from the gathered Maven metadata. Due to these reasons, we have excluded Maven from certain aspects of analyses, which has little impact on the findings of our study. Table 1 shows the overall statistics for the five platforms.

2.2 License Extraction

In this study, we focus on versioned packages with license information. The metadata of all five platforms contain corresponding *license* fields for extracting license information. In addition, we conduct further searches from other fields that may contain licenses such as *classifier* field in PyPI as a supplement. Please refer to our replication package [14] for specific search methods.

Meanwhile, the provided license names from metadata may not always follow the standard format or identifiers listed in the SPDX license list, sometimes even being hidden within a block of text. For instance, the package *multipy==0.16* in PyPI has a *license* field with the content "Revised 3-clause BSD" which corresponds to the SPDX license with the full name "BSD 3-Clause 'New' or 'Revised' License" and the identifier "BSD-3-Clause".

However, some off-the-shelf tools, such as Lidetector [50] and ScanCode [15], necessitate specific clauses or full license identifiers as input, which is not readily available in metadata information. Since metadata lacks license term details, we propose an algorithm to extract license information from metadata, which is as follows:
Building a license pool: We first build a license pool that contains all the SPDX licenses we collected. The license pool contains the standard full names and identifiers of 469 licenses in the SPDX License List [38] for version 3.20.

Our license extraction algorithm is designed to search for licenses from a pile of complex texts (e.g., the *classifier* field of PyPI in Figure 1b) and map non-standard customary full names or identifiers of licenses to their standardized formats in license pool.
Extracting licenses: We implement the extraction algorithm based on the AC (Aho-Corasick) automaton [17] and regular expression matching. AC automaton, an efficient multi-pattern string matching

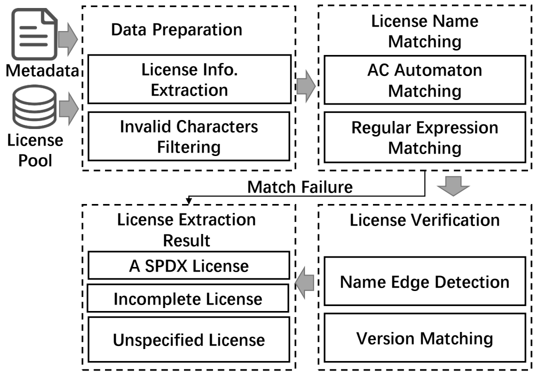


Figure 2: Overview of License Extraction.

algorithm, can match multiple pattern strings in a large text string at the same time.

Given a textual description, our algorithm uses the AC automaton and regular expression to match licenses. More specifically, our algorithm inputs names of all SPDX licenses in the license pool into an AC automaton. For each SPDX license, it is divided into two parts: the name and the version, and all versions of the same name would be aggregated into a set. For example, the version set of the Mozilla Public License has four versions, i.e., 1.0, 1.1, 2.0, and 2.0-no-copyleft-exception. If the licence has no version, the version set is empty. For the matched results, we conduct edge detection which aims to check if the match is part of other words to avoid false positives, e.g., detecting "MIT" from "com**mit**". If a license name is successfully matched, the version is matched in subsequent content.

In addition, we use regular expressions as a supplementary aid to accurately extract licenses, especially for abbreviated formats and those including subsequent versions. For example, in the case of "AGPLv3+", the AC automaton cannot pass "edge detection" and cannot extract "+". After extracting the possible license name, it is still necessary to enter this name into the AC automaton to confirm that it is indeed the name of a particular SPDX license, and the possible version also needs to be detected in the corresponding version set.

Our license extraction algorithm has three outcomes, i.e., an *SPDX license identifier*, *incomplete* (unspecified version or unavailable version), or *unspecified* (empty input or failing to match a license in the pool). For example, for an input string "Licensed under the LGPLv3+", our approach can extract the license name as 'LGPL' and the version as '3.0+', indicating that the version can be later than 3.0. So, the result is "LGPL-3.0-or-later". In addition, We use the term *license series* to describe incomplete licenses, such as the BSD series.

In the package metadata, multiple licenses may be declared simultaneously. For instance, in PyPI, *pygatt==4.0.4* declares its license as "Apache 2.0 and MIT". Our license extraction algorithm attempts to search for all existing licenses and stores them in a string separated by a delimiter.

To test the capability of the extraction algorithm, we randomly selected 20,000 non-empty license records per platform, resulting in a total of 100,000 license records. After removing duplicates, we

Table 2: Extraction Failure Rate of License Extraction.

Platform	% Extraction Failure
Maven	**1.17**
NPM	**3.17**
PyPI	1.55
RubyGems	1.99
Cargo	2.59

were left with 1,407 license information records. We manually annotated these 1,407 license information and then inputted them into the extraction algorithm. The verification was manually conducted by the first author after sampling the metadata. Given the input length was generally not more than 100 characters and the high similarity, manual identification was not overly challenging. We verified a total of 100,000 data points, which took approximately 4 hours. After verification, the extraction algorithm achieves 97.14% accuracy and 95.24% recall for *SPDX* license names, and 95.56% accuracy and 88.11% recall for *incomplete* license names. This shows that our license extraction algorithm has high reliability.

2.3 Incompatibility Detection Algorithm

In this study, we want to investigate the license compatibility issues in the selected platforms. To improve the credibility of the results and to show the results under different license incompatibility definitions, we used two tools for testing. License-compatibility [29] maintains that two licenses can be considered compatible as long as there are no explicit conflicts. On the other hand, Lidetector [50] argues that compatibility should encompass broader factors, such as the requirements and restrictions of the licenses, as well as the consistency in meeting each term's conditions, which are as follows:

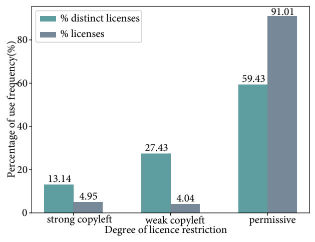
- **LiDetector** leverages a learning-based approach to automatically identify meaningful license terms from an arbitrary license text [50]. Then, it applies Probabilistic Context-Free Grammar (PCFG) to infer the attitude of the current license text with respect to the rights and obligations in the license terms. Finally, it detects license incompatibilities by comparing the attitude of all licenses with respect to the same terms.
- **License-compatibility** is an OSS tool from Libraries.io [29], a software versioned package indexing platform, that provides a license compatibility matrix, along with semantic analysis. The tool uses metadata from OSS licenses and the SPDX standard to detect compatibility between two or more licenses.

The differing methodologies indeed yield varying results for the same example. For instance, Lidetector deems Apache-2.0 incompatible with MIT due to Apache-2.0's additional obligations, potentially causing MIT compliers to violate Apache-2.0. Conversely, License-compatibility considers Apache-2.0 compatible with MIT. Both perspectives are valid based on their respective definitions of incompatibility, and we aim to examine this from both angles.

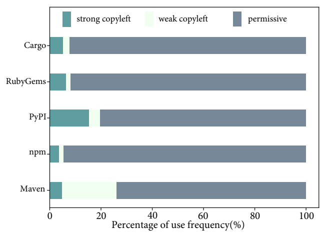
We use a dataset provided by the LiDetector tool as a test set to check the capability of LiDetector and License-compatibility. This dataset from [50] consists of 200 projects randomly selected from 1,846 popular GitHub projects with more than 1,000 stars and has been manually validated and cross-validated by authors

Table 3: Statistics of license usage on different platforms.

	Maven	NPM	PyPI	RubyGems	Cargo
%SPDX License	**28.71**	85.73	71.16	64.34	**97.27**
%Unspecified License	**56.38**	13.8	15.41	34.61	**2.73**
%Incomplete License	**14.91**	0.47	13.43	1.05	**0.0009**
%Multi-licenses	1.15	**0.81**	4.14	0.46	**28.39**



(a) Ratio and Usage of Licenses.



(b) Percentage on five platforms.

Figure 3: The usage of different types of licenses.

and lawyers. After testing, we find that LiDetector achieves a precision of 81.46%, a recall of 78.85%, and an F1-score of 0.8, while License-compatibility which only detects SPDX license in dataset achieves a precision of 73.55%, a recall of 79.17%, and an F1-score of 0.76. After testing, both of these tools have a good ability to detect incompatibilities.

3 EXPERIMENT RESULTS

In order to comprehensively investigate license data among the five package management platforms, we formulate and address the following five research questions:

- **RQ1.** What is the status quo of license usage among five package management platforms, and what are their similarities and differences?
- **RQ2.** How often do the licenses change in the version updates of the package? How does it change?
- **RQ3.** How common is the issue of license incompatibility between a versioned package and its direct dependencies, and what are the main terms that cause such incompatibilities?
- **RQ4.** What is the temporal evolution in the use of licenses? Which licenses are more popular?
- **RQ5.** What are the differences in license usage between the overall packages and the more popular packages, as well as among the popular versioned packages on five platforms compared horizontally?

3.1 RQ1 – License Usage

Motivation and Approach. In the initial phase of our research, we seek to gain a comprehensive understanding of the overall status of packages on package management platforms, which enable us to quickly discern the characteristics and differences among packages on these platforms.

We count the percentage of versioned packages with *SPDX*, *Unspecified* and *Incomplete* licenses. Furthermore, we classify versioned packages with *SPDX* licenses into *single-license* and *multi-license* versioned packages since the versioned packages with *multi-licenses* tend to have more complex licensing terms and are more likely to pose legal risks. We are also interested in licenses with different degrees of authorization restrictions. Hence, we use commonly used criteria to divide licenses into three categories, i.e., *strong copyleft*, *weak copyleft*, and *permissive*.

Strong copyleft licenses require derivative works or modifications of the licensed software to be distributed under the same copyleft license terms. Weak copyleft licenses require derivative works or modifications of the licensed software to be distributed under compatible copyleft license terms. Permissive licenses allow users to incorporate the licensed software into proprietary or closed-source projects without being required to release the source code of those projects. We used license category data from TLDR-Legal [16], a well-known website that provides brief summaries of open source software licenses.

Finding 1-1*: There is a high proportion of versioned packages with unspecified or incomplete licenses.*

Table 3 presents the number of versioned packages and the percentage of versioned packages with *SPDX licenses*, *unspecified licenses*, *incomplete licenses*, and *multi-licenses*. As shown in the table, *unspecified* licenses exist more or less in the versioned packages from all five platforms. The main component for *unspecified* licenses is the lack of a license. For example, 55.09% of versioned packages in Maven lack licenses. This is because Maven does not require specifying license information in the POM file. The licenses for these libraries may have relevant information elsewhere but are not included in the metadata we collect. On the other hand, Cargo has the lowest proportion of *unspecified* licenses (2.73%). The other three platforms also have a significant number of versioned packages with *unspecified* licenses, posing a high risk of legal disputes.

The percentage of *incomplete* licenses in Maven and PyPI is significantly higher than in the other three platforms, which do not exceed 2%. Specifically, Maven has 14.91% *incomplete* licenses, while PyPI has 13.43%. Although an *incomplete* license provides information about the type of license in comparison to a *unspecified* license, there still exists considerable legal risk due to the uncertainty of the specific version and terms of the license.

For the versioned packages with *multi-licenses*, Cargo has the highest proportion (i.e., 28.39%), which far exceeds the other four platforms. We find a clear dominant license combination in *multi-licenses* in NPM and Cargo. In NPM, the combination of EPL-2.0 and GPL-2.0-only accounts for 64.84%, while in Cargo, the combination of MIT and Apache-2.0 accounts for 96.03%. It's worth noting that the combination of EPL-2.0 and GPL-2.0-only is conditionally compatible, while MIT and Apache-2.0 are compatible. Cargo puts a lot of emphasis on software license selection and normalization, while MIT and Apache-2.0, two of the most popular licenses, are compatible and complementary in their combination and are therefore chosen by most Cargo's versioned packages.

Finding 1-2*: Permissive licenses have a high usage percentage of over 90%. GPL-3.0-only, MPL-2.0, and MIT are the most commonly used licenses for each of the three levels of restriction, respectively.*

Figure 3 shows the statistics on the usage of licenses with different types of restrictions. We find a total of 236 distinct SPDX licenses in our data, including 23 *strong copyleft* licenses, 48 *weak copyleft* licenses, 104 *permissive* licenses and 61 licenses with unclearly defined restrictions. We ignore the 61 unclassified licenses as they constitute only 0.21% of all packages. Although the number of *permissive* licenses is only 104, accounting for 59.43% of all licenses classified, its usage rate is very high, making up 91.01% across the five platforms. The usage rates of the two copyleft licenses are less than 5%.

Table 4 shows detailed statistics on the specific use of the three types of licences within the scope of SPDX licences in our aggregated data for the five platforms. Among the *strong copyleft* licenses, GPL family licenses appear very frequently, and the most used one is the GPL-3.0-only license, with 37.61% usage rate. Among the *weak copyleft* licenses, the most used one is the MPL-2.0 license with 23.98% usage. Among the *permissive* license, the MIT license is the most widely used among permissive licenses, with a usage rate of 61.80%. The utilization of different licenses stems from the fact that permissive and copyleft licenses have their own strengths and weaknesses, and their selection often hinges on the specific objectives pursued.

Finding 1-3*: Cross-platform comparisons show some commonalities and differences in the use of licenses.*

We conduct a comparative analysis of license usage across different package management platforms. Similarities in license usage are observed among the various platforms. As depicted in Table 5, the combined usage rate of the MIT license and the Apache-2.0 license exceeds 70% in all package management platforms, and even surpass 85% in RubyGems and Cargo. Among the SPDX licenses of the four platforms, both licenses are in the top two in usage, while in the SPDX licenses of NPM, the MIT license is in the first place and the ISC license squeezes the Apache-2.0 license into third place.

However, some variations in license preference are also observed among the different package management platforms. Regarding the preference for license types among the five platforms, two points are particularly prominent. First, the *weak copyleft* license of Maven has a high usage rate of 9.59%, with LGPL-2.1-only accounting for 5.67%, while the other four platforms' *weak copyleft* licenses are used at less than 5%. Second, PyPI has a significantly higher usage proportion of *strong copyleft* licenses, which is more than twice that of the second-place platform.

In NPM, the ISC license breaks the dominance of the MIT and Apache-2.0 licenses and ranks second in usage rate. However, the ISC license is not commonly seen in other package management platforms and is based on the BSD 2-term license, with only three lines of content, making it a very permissive license.

3.2 RQ2 – License Change

Motivation and Approach. License changes for open source software can have significant impacts on both the software project and its users [37, 43]. Additionally, license changes can cause compatibility issues with other software or versioned packages [25, 45, 48], particularly if the new license is not compatible with existing licenses in use. In this RQ, we examine cases where licenses are

Table 4: The top five most popular licenses of the three types of licenses in all data.

Rank	Strong Copyleft	Weak Copyleft	Permissive
1	GPL-3.0-only (37.61%)	MPL-2.0 (23.98%)	MIT (61.80%)
2	AGPL-3.0-only (17.51%)	LGPL-2.1-only (21.41%)	Apache-2.0 (19.89%)
3	GPL-2.0-only (15.60%)	LGPL-3.0-only (19.71%)	ISC (15.47%)
4	GPL-3.0-or-later (11.42%)	EPL-2.0 (17.61%)	BSD-3-Clause (1.61%)
5	GPL-2.0-or-later (7.74%)	LGPL-3.0-or-later (3.26%)	BSD-2-Clause (0.46%)

Table 5: The top five most popular licenses among the five platforms.

Rank	Maven	NPM	PyPI	RubyGems	Cargo
1	Apache-2.0 (59.42%)	MIT (60.27%)	MIT (54.64%)	MIT (70.51%)	MIT (51.22%)
2	MIT (23.78%)	ISC (18.66%)	Apache-2.0 (20.03%)	Apache-2.0 (18.25%)	Apache-2.0 (36.10%)
3	LGPL-2.1-only (5.66%)	Apache-2.0 (13.22%)	GPL-3.0-only (6.14%)	GPL-2.0-only (2.72%)	GPL-3.0-only (2.45%)
4	GPL-3.0-only (2.97%)	BSD-3-Cause (1.41%)	AGPL-3.0-only (3.79%)	GPL-3.0-only (1.78%)	MPL-2.0 (1.69%)
5	BSD-3-Clause (1.33%)	GPL-3.0-only (1.14%)	GPL-3.0-or-later (2.34%)	BSD-3-Cause (1.09%)	BSD-3-Cause (1.20%)

Table 6: Licence changes on different platforms.

Platform	#Modification	#Addition	#Deletion	%Change
Maven	38,161	14,611	10,091	0.64%
NPM	156,315	45,015	16,151	0.79%
PyPI	44,063	10,616	4,217	1.50%
RubyGems	14,544	10,765	962	2.12%
Cargo	7,688	844	31	1.41%

Table 7: Changes between Single and Multiple Licenses.

Platform	# S→M	# M→S	# M→M	# S→S
Maven	596 (1.7%)	404 (1.2%)	234 (0.7%)	33,934 (96.5%)
NPM	8,388 (5.9%)	8,047 (5.6%)	75 (0.1%)	126,913 (88.5%)
PyPI	3,940 (11.5%)	3,849 (11.2%)	**633 (1.8%)**	25,978 (75.5%)
RubyGems	287 (2.1%)	61 (0.4%)	53 (0.4%)	**13,445 (97.1%)**
Cargo	**2,249 (37.8%)**	**784 (13.2%)**	104 (1.8%)	2,810 (47.3%)

S means single-license, M means multiple-licenses

Table 8: Changes of License Restrictiveness.

Platform	#More Permissive	#More Restrictive	#No Change
Maven	1,474 (15.8%)	753 (8.1%)	7,106 (76.0%)
NPM	4,439 (7.2%)	4,988 (8.1%)	**52,350 (84.7%)**
PyPI	2,504 (22.0%)	1,707 (15.0%)	7,179 (63.0%)
RubyGems	467 (25.1%)	328 (17.6%)	1,066 (57.3%)
Cargo	**547 (26.4%)**	**501 (24.2%)**	1,026 (49.5%)

changed in versioned packages, including changes in the number of licenses and the degree of restrictions.

***Finding 2-1**: License changes are rare. The percentages of license changes are slightly higher in PyPI, RubyGems, and Cargo.*

Table 6 presents the number and percentage of different types of license changes across five platforms. We divide license changes into three types, i.e. *Addition* (a license is newly added in a package), *Deletion* (a license is removed from a package), and *Modification* (the license of a package is changed). We find that license changes are relatively rare between the updates of packages (i.e., only 0.86%). In fact, the percentage of license changes will be higher because some developers may change the source code without indicating whether the change involves the license header [28]. PyPI, RubyGems, and Cargo have a slightly higher proportion of license changes, but the data volume for these three package management platforms is relatively small.

***Finding 2-2**: The transitions between single and multi-licenses in Cargo and PyPI are more frequent than in the other platforms.*

Table 7 shows the number of license changes in terms of single and multiple licenses. There are four types of changes in the table. The majority of the changes are from single license to single license. As shown in the table, the data in Cargo is significantly different from the data in the other platforms, with 37.82% of the changes from single to multiple licenses. 90.52% of these multi-license changes are a combination of MIT and Apache-2.0. These two licenses are also the most targeted choices (72.64%) when changing from multiple licenses to a single license. In addition, the transitions between single and multiple licenses in PyPI exceed the average, with both accounting for over 10% of the total.

We observe that most addition-type changes involve adding new licenses on top of the existing ones without altering the original licenses. Similarly, the majority of deletion-type changes remove a portion of the original licenses while retaining another part. However, this is not the case in NPM. our statistical analysis shows that 91.18% of license changes in NPM involve transitions between Apache-2.0 and a combination of GPL-2.0-only and EPL-2.0, which is an exception to the general pattern we observe.

***Finding 2-3**: The degree of restriction of licenses changes less in NPM and more in Cargo.*

As shown in Table 8, we find that there are considerable numbers of license changes with different restrictions. The direction of license changes of NPM is relatively the most stable, with 84.74% of license changes being made under equally restrictive licenses. On the other hand, Cargo has the highest percentage of changes in both the more restrictive and more permissive directions, indicating a wider range of changes in terms when changing licenses. We find that the most frequent combination of license changes that changes the degree of restriction in Cargo is Apache-2.0 and

Table 9: Incompatibility detection results for overall versioned packages.

Tools	Maven	NPM	PyPI	RubyGems	Cargo
LiDetector	7.33%	4.14%	**3.29%**	7.17%	**7.81%**
License-Compatibility	6.33%	3.60%	**3.04%**	6.71%	**7.72%**

GPL-3.0. In fact, as the two popular licenses, Apache-2.0 is considered forward-compatible with GPL-3.0, and the community of developers emphasizes open source and licensing norms, leading to frequent consideration of license changes and conversions.

3.3 RQ3 – License Incompatibility

Motivation and Approach. Not all open source software licenses are compatible with each other [42, 47, 49]. Incompatible licenses may lead to legal issues and potential conflicts between different open source projects [35, 44]. In this RQ, we want to investigate the license incompatibility in versioned packages. Given the technical difficulties of detecting license incompatibilities in passing dependencies, we will only consider license incompatibilities in direct dependencies.

Given a versioned package in our dataset, we can get its direct dependencies in its metadata information. For example, each record in the PyPI metadata has a *dependencies* field. For range-based dependency requirements, we select the latest package version that meets the requirement. Thus, we can get a license set by combining the versioned package license and the licenses of its direct dependencies. Then, we use the license incompatibility detection tools in Section 2.3 to identify compatibility issues between each pair of licenses in the set. Furthermore, the LiDetector algorithm can extract license terms and analyze attitudes toward them. So, we can obtain more detailed information about specific terms that cause incompatibility results. Note that we exclude the versioned packages without direct dependencies or SPDX licenses in this analysis.

Finding 3-1*: The incompatibility rate of versioned packages for all five platforms is below 8%, with the highest incompatibility rate in Cargo and the lowest incompatibility rate in PyPI.*

Table 9 shows the license incompatibility ratios detected by the two tools across five platforms. LiDetector detected more compatibility issues than License-compatibility. We manually examine the inconsistent results between the two tools and find that the incompatibilities detected by License-compatibility are basically covered by LiDetector. The two main reasons are ❶ License-compatibility does not cover some of the latest licenses added to the SPDX license list, e.g., BSL-1.0. ❷ Lidetector has a stricter definition of license incompatibility. For example, License-compatibility considers that there is no incompatibility between MPL-2.0 and GPL-3.0, but Lidetector disagrees.

As shown in Table 9, the proportion of license incompatibility in Maven, RubyGems and Cargo is much higher than that in NPM and PyPI. The incompatibility rate in Cargo is the highest, with an average result of 7.77%, and the incompatibility rate in PyPI is the lowest, with an average result of 3.17%. The highest incompatibility rate does not exceed 8%, indicating that the data in the package management platform is relatively more standardized compared to other studies conducted on Github projects [47, 50].

Table 10: The five terms that cause the most incompatibilities on different platforms.

Terms	Maven	NPM	PyPI	RubyGems	Cargo
%Sublicense	82.95	56.21	44.31	84.37	40.56
%Hold Liable	4.16	11.61	15.46	6.85	16.83
%Distribute	3.87	11.19	15.21	5.26	16.29
%Place Warranty	2.91	6.16	9.89	2.13	14.77
%Commercial Use	1.74	4.21	4.54	0.66	4.57

Finding 3-2*: The most common type of license incompatibility arises from combining permissive licenses with GPL family licenses.*

LiDetector categorizes license terms based on tldrlegal [16], which is a platform that provides simplified versions of well-known license terms. There are 23 types of terms, including 11 right terms and 12 obligation terms. Table 10 shows the five terms that cause the most license incompatibilities among all the data.

The *sublicense* term describes the ability of developers to grant/extend a license to the software; and the *hold liable* term describes the warranty and if the software/license owner can be charged for damages; and the *distribute* term describes the ability to distribute original or modified (derivative) works; and the *Place Warranty* term describes the ability to place warranty on the software licensed; and the *Commercial Use* term describes the ability to use the software for commercial purposes.

These terms, which encompass the most significant concerns for users when using OSS, are all right terms that involve the authorization of the user or licensee and protect most of the rights of the holder. The most frequent incompatible term across five platforms is *Sublicense*, which is far ahead of other terms. Most incompatibilities are caused by the mixture of permissive licenses and GPL family licenses.

3.4 RQ4 – Temporal Evolution

Motivation and Approach. As the open-source community continues to grow and expand, new licenses are introduced every year, each representing a collection of new terms [46]. At the same time, some licenses are becoming less popular over time and may eventually disappear. The emergence and disappearance of licenses can in some sense reflect the evolution of open-source software, so we pay attention to the usage of licenses on package management platforms in different periods.

Based on the metadata we extract from the five platforms, we are unable to locate the release time information for versioned packages in Maven. However, it is possible that this information is available in other sources. So in this RQ, we focus on the other four platforms. The earliest release dates on the four platforms are different, with the earliest release information appearing in NPM, PyPI, RubyGems, and Cargo in 2010, 2005, 2009, and 2014, respectively.

By utilizing the temporal information, we analyze the overall usage and annual changes of licenses on each platform. For a specific time period, we analyze the most popular licenses during that period. For a specific license, we analyze its earliest appearance time and latest usage time on the platform to determine whether it becomes more popular over time.

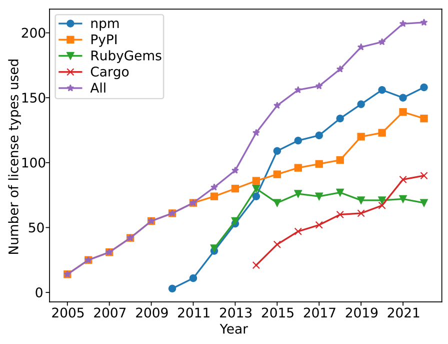


Figure 4: Number of distinct licenses used each year.

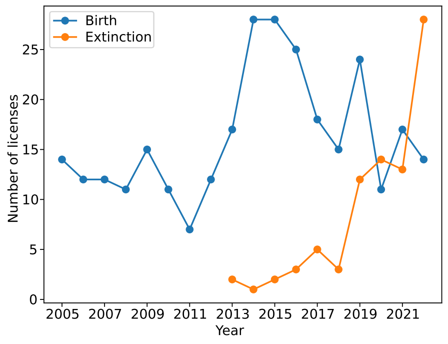


Figure 5: Number of new and extinct licenses.

> ***Finding 4-1****: The annual growth rate of the number of distinct licenses is decreasing. The gap in quantity between newly emerging and extinct licenses is gradually narrowing every year.*

Figure 4 shows the number of distinct licenses used each year. During the five-year period from 2011 to 2016, the number of licenses increased essentially linearly, with an average increase of more than 20 licenses per year. This indicates the booming development of the open source software. After 2016, the increase in OSS licenses began to slow down. Existing licenses already meet most of the needs of open source software.

We consider the year of a license's birth as its first appearance and the year of a license's extinction if no packages use it during a year (except 2022). Figure 5 shows the annual number of new and extinct licenses. The birth and extinction of licenses can reflect the trend of license evolution in the field of open-source software. We find that in recent years the number of new licenses is fluctuating down, while the number of extinct licenses is fluctuating up. Some licenses with unsuitable terms are gradually falling out of use, and more attention is being paid to popular licenses whose terms are designed to meet the needs of the people. After years of evolution, the currently popular licenses largely meet the various term requirements of users. The trend of licenses is to increase the reuse rate of each license rather than freely increasing their numbers. This highlights the importance of designing licenses that are flexible and can be widely adopted.

> ***Finding 4-2****: Apache-2.0 is trending to gradually replace MIT as the most popular license.*

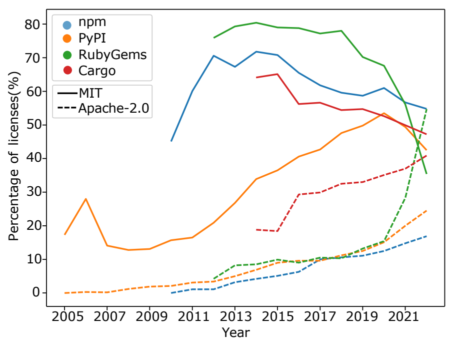
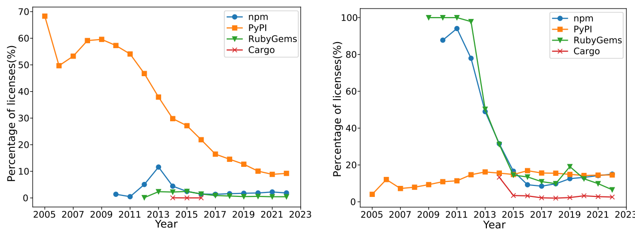


Figure 6: Evolution of MIT and Apache-2.0.



(a) Incomplete licenses. **(b) Unspecified licenses.**

Figure 7: Evolution of the proportion of incomplete licenses and unspecified licenses over time.

We collate the most popular batch of licenses for each platform during each period. The top three in NPM are MIT, ISC and Apache-2.0, and the share of MIT and ISC decreases year by year, while the share of Apache-2.0 increases year by year. In the history of PyPI, the GPL and BSD series of licences used to have a high percentage, but in recent years they have been gradually dominated by MIT and Apache-2.0, and in 2022 the top three licences in terms of popularity are MIT, Apache-2.0 and GPL-3.0-only. There is very little change in the popularity of licences in RubyGems and Cargo. In RubyGems, Apache-2.0 became the most popular licence in 2022, and before that the top three were basically MIT, Apache-2.0 and GPL-2.0-only. While in Cargo, MIT and Apache-2.0 have been the most popular licences, with GPL-3.0-only replacing MPL-2.0 as the third most popular licence in 2018. We document the evolution of MIT and Apache-2.0 in different platforms in Figure 6. MIT and Apache-2.0 are both highly popular across all four platforms in recent years. but Apache-2.0 is trending to gradually replace MIT as the most popular license.

> ***Finding 4-3****: The incomplete licenses issue is improving year by year in all four platforms. In NPM and PyPI, the issue of unspecified licenses is still severe with no signs of improvement.*

Figure 7 presents the trends of usage of incomplete licenses and unspecified licenses over time. The issue of incomplete licenses has been improving over time in four platforms. It has remained low in proportion in the other three platforms except PyPI and disappeared completely in Cargo after 2016. There is still a not-insignificant percentage of incomplete licenses in PyPI, with 9.23% in 2022.

On the issue of unspecified licenses, the situations are getting better in both RubyGems and Cargo, but they have never been improved in PyPI, which has remained around 15%. And in NPM, the issues continued to improve until 2017, but became even more severe after 2017.

After further analysis, we find that the reason for the high percentage of incomplete licenses in PyPI during the initial stages is that most versioned packages use GPL or BSD series licenses. The reason for the high percentage of unspecified licenses in NPM and RubyGems during the initial stages is that most versioned packages do not declare a license which is also the main reason why the problem of unspecified licenses in NPM got worse after 2017.

We also observe that the proportion of multiple licenses in Cargo has gradually decreased in recent years, indicating that license management in Cargo is also improving. As a smaller and shorter-developed package management platform, the data from Cargo is relatively well-distributed in the overall dataset. In addition, we conduct a statistical analysis of license changes over time, and find that the proportion of license changes on all four platforms has been decreasing year by year in recent years, tending to be consistent, with an average of about 0.4% in 2022.

3.5 RQ5 – Core Packages

Motivation and Approach. In the OSS community, some packages are particularly popular and are often depended on by many other packages. Such core packages usually have a high impact on the packages that depend on them [31]. Therefore, we want to explore the characteristics of these highly popular versioned packages.

For each platform, we construct a weighted directed graph based on the dependencies among packages. In the dependency graph, a node presents a package. If there exists one version of a package A depends on one version of another package B, we construct an edge from A to B. Based on the constructed dependency graph, we use the PageRank [20] algorithm to evaluate the influence of a package. The more important a package is, the larger its PageRank value. We consider a package as a core package if its PageRank value belongs to the top 5%. Finally, we obtain *546*, *2,177*, *274*, *390* and *440* core packages for Maven, NPM, PyPI, RubyGems, and Cargo, respectively.

Finding 5-1: *The license information of the core packages in RubyGems is less standardized, while that in Maven and PyPI is more standardized.*

Table 11 shows the license usage of core versioned packages of each platform. 87.84% of Maven's core versioned package licenses are unspecified, possibly because Maven does not require that the `POM` file from which we collect information must have a license field. The proportion of the *unspecified* licenses of the core versioned packages in RubyGems has increased from 3.17% to 15.04%, which is more non-compliant. Additionally, the percentage of *incomplete* license names in the core versioned packages in Maven and PyPI decreases by approximately 5%, and the percentage of *incomplete* licenses in RubyGems and Cargo remained much lower than the other three platforms, indicating a greater level of standardization in the data. With respect to versioned packages with multiple licenses, the proportion in Cargo increases from 23.39% to 48.42%, whereas the proportion for other platforms, whether in core or overall versioned packages, does not exceed 5%, due to a higher

Table 11: Statistics on license usage of core versioned packages.

	Maven	NPM	PyPI	RubyGems	Cargo
%Unspecified License	87.84	10.38	14.21	36.70	0.82
%Incomplete License	6.91	4.40	9.85	0.48	0.01
%Multi-License	0.90	4.74	1.45	0.85	**48.42**
%Strong Copyleft License	1.66	8.02	11.28	**30.82**	0.02
%Weak Copyleft License	24.12	4.92	9.10	5.37	0.94
%Permissive license	74.21	87.06	79.62	63.81	**99.04**

Table 12: Incompatibility detection results for core versioned packages.

Tools	Maven	NPM	PyPI	RubyGems	Cargo
LiDetector	8.44%	9.00%	3.99%	15.83%	0.25%
License-compatibility	7.48%	7.35%	3.48%	10.73%	0.23%

proportion of versioned packages that use both MIT and Apache-2.0 licenses in Cargo's core versioned packages.

Finding 5-2: *The use of strong copyleft licenses is significantly higher in RubyGems's core versioned package, and significantly lower in Cargo's.*

Regarding license restriction types, the proportion of strong copyleft licenses in RubyGems' core versioned packages increases from 6.36% to 30.82%, more than twice as much as the second place PyPI. More than 90% of licenses of many core packages are GPL-2.0-only, such as `rcfiles`, `rbt`, and `roebe` in RubyGems. Conversely, the proportion of permissive licenses in Cargo's core versioned packages rises to 99.04%. In Cargo's core package, only two copyleft licenses, GPL-3.0-only and MPL-2.0, are used slightly, and more than 96% of the core repositories are using Apache-2.0 or MIT. This result suggests that RubyGems' core versioned package places more emphasis on collaboration and sharing in the open source community, while in Cargo there is a preference for a free approach to distributing software.

We have observed license alterations within core packages. Specifically, the versioned packages for Maven, NPM, PYPI, RubyGems, and Cargo exhibited license changes at rates of 0.55%, 0.81%, 0.14%, 0.12%, and 0.63% respectively. Notably, most platforms demonstrated lower figures in comparison to the overall packages. Given the limited number of results (Cargo had the fewest with only 219 results), we did not delve further into this aspect.

Finding 5-3: *The percentage of incompatible licenses for Cargo's core package decreases significantly, while it increases in all other four platforms.*

We test for incompatibilities in the core versioned packages and documented them in Table 12. Compared to the previous data, the results of the two tools show that license incompatibilities are occurring more in four platforms except Cargo. The percentage of license incompatibilities has increased slightly in Maven and PyPI, and are nearly twice as much in NPM and RubyGems as before. In contrast, the phenomenon is most striking in Cargo, where the percentage of incompatibilities in the core package drops very sharply, approximately one thirtieth of the original. This indicates that Cargo's core versioned package manages licenses very well and has very low incompatibilities. In addition, *Sublicense* is still the most incompatible term, and the mix of components under the permissive license and copyleft license is still the most important point of concern in core versioned packages.

4 DISCUSSION

In this section, we first discuss several implications for action based on the findings in our study, then describe the threats to validity.

4.1 Implications

Unspecified and incomplete licenses have a significant impact on the license specification in the package management platform. There is a high proportion of versioned packages with unspecified or incomplete licenses (*Finding 1-1*). The lack of licenses is a major aspect of unspecified licenses and a more serious issue than incomplete licenses because packages that lack explicit licenses are typically assumed to retain all rights, which can exacerbate license incompatibility problems. Fortunately, we checked that the incomplete licenses issue is improving year by year in all four platforms, but the issue of unspecified licenses in NPM and PyPI is still severe (*Finding 4-3*), which deserves further exploration by researchers. For developers, using the license recommendation tool [27] to select licenses for new projects or older projects with missing licenses is a good option to reduce the legal risk. For the platform, a reasonable screening of released packages can effectively reduce the proportion of legal hazards. The issue of license irregularities in package management platforms needs more attention because developers often rely on platforms to make their dependencies work easily and quickly.

Irregularities in the licensing of core libraries need attention. In RubyGems' core package, the proportion of the unspecified licenses increases significantly (*Finding 5-1*). Given that the impact of the core package is much greater than that of the overall packages, this needs to be taken into account. Incompatibility test results for the core package show an increase in incompatibility percentages for all platforms except Cargo (*Finding 5-3*). As the bottom of the dependency chain, every update of the core package has a profound impact, and its license specification needs to be guaranteed. Developers of core packages can use license incompatibility detection tools to detect the currently selected license before releasing a new version of the package, and the development of better incompatibility detection tools is a direction that needs to be explored continuously, for example, a large language model [21, 39, 40] has good text analysis capabilities and potential.

License changes are one of the most likely aspects of license incompatibility. On the one hand, developers tend to use historical versions of licenses [41], but their dependency packages may have changed; on the other hand, license changes often occur with a large span of changing license restriction types (*Finding 2-2*) and conversion from single to multiple licenses (*Finding 2-3*), which can easily lead to license incompatibility. For developers who are not sure of the package license, we recommend choosing a permissive license such as MIT to reduce the risk. Our research shows that more than 90% of the licenses in the package management platform are permissive licenses (*Finding 1-2*), and it is more likely to create hidden problems if the GPL family licenses are chosen (*Finding 3-2*).

Community guidance is important for the development of licenses in package management platforms. Cargo, the youngest of the package management platforms we studied, has the best overall license performance in our statistics, with not only the lowest percentage of unspecified and incomplete licenses (*Finding 1-1*), but also the lowest incompatibility rate of core packages (*Finding 5-3*). Rust, as the official language of Cargo, has always been known for its high security, ensuring less technical debt and more standardized license management in the Rust community. Other platforms often overlook the importance of license specifications due to their lengthy development process. This negligence leads to the accumulation of license defects, which are further exacerbated by the interdependence of packages, thereby posing a significant hidden risk. It requires developers to be more aware of license regulations, as well as proper guidance from the authorities to create a more regulated community atmosphere.

The trend of license evolution is important for researchers. Although the number of new packages added each year is very high, making the use of licenses more widespread, the trend of increasing the variety of licenses is slowing down (*Finding 4-1*). With unpopular licenses being phased out and popular licenses becoming more widely used, analyzing the difference between the two is a meaningful direction for research. We observe that Apache-2.0 is becoming more popular than MIT license(*Finding 4-2*), possibly due to its compatibility with GPL-3.0. Developers can switch between these licenses based on user obligations. Similarly, Licenses with similar rights or obligations can serve as alternatives. Hence, a study on license change or selection is valuable.

4.2 Threats to Validity

Internal validity. There are quite a few versioned packages with missing licenses. For example, Maven has a high percentage of missing licenses, possibly due to our data collection method. We tested the SPDX licenses of package management platforms and selected five representative ones with a high number of SPDX licenses. However, the license declaration of versioned packages may contain noisy information, such as irregular names of licenses. Our high-precision license extraction algorithm demonstrates reliable handling of such cases, with iterative improvements based on the dataset to accommodate additional license declaration scenarios. Another threat is the results of license incompatibility detection. Our license incompatibilities only focus on direct dependencies, but according to recent study [51], there are also high levels of license incompatibilities in indirect dependencies, so the true level of license incompatibilities in package management platforms is higher than our results. The two tools represent two incompatible definitions, while also complementing each other's lack of design logic and precision. LiDetector may misidentify license terms as it can only handle predefined types, while License-compatibility may not process newer licenses. Therefore, more tools with good text analysis capabilities can be taken a step further, such as large language models.

External validity. The development time and popularity of different package management platforms vary greatly, and our statistical and discovery findings for the current platform may not be generalizable to other platforms. To mitigate this threat, we carefully select data from five platforms. Among them, Maven, NPM, and PyPI are the most popular and widely used platforms, with a large versioned package network and corresponding programming languages that are also the most popular languages on GitHub and used in various fields. RubyGems and Cargo have relatively fewer users and versioned packages, but they have also gained a lot of popularity in recent years. By selecting with different versioned package sizes, we make our results more widely applicable. We believe that the study of licenses for these five platforms can provide different insights for open-source developers from multiple perspectives.

5 RELATED WORK

We summarize related work from these perspectives:

License Incompatibility. Qiu et al. [36], conducting an empirical study of NPM, found that including packages licensed under a copyleft license in a dependency network may result in highly dependent-related licensing violations. Similarly, Makari et al. [30], who studied the evolution, prevalence, and compliance of dependent licenses in the NPM and RubyGems package ecosystems, found that GPL dependencies were a major cause of incompatibility. Their findings are consistent with our summary of the license's incompatible provisions. In addition, Makari et al. [30], found that 3.1% of NPM and 9.9% of RubyGems dependency packages were affected by license incompatibility. The huge contrast between the license incompatibilities of packages within the package management platform and Github projects is something worth investigating.

License Usage. Moraes et al. [34], and Almeida et al. [18] both surveyed developers, and their results consistently showed that while developers were aware of the use of one license, they were not very good at understanding multiple licenses. Meloca et al. [33] observed that about 24% of the released packages used at least one of the hundreds of non-OSI-approved licenses they detected, most of which were lack of license. They also found that package publishers preferred to use the same licenses that had been used on historical releases, which is consistent with our Finding 2-1 that license changes occur rarely.

License Changes. Vendome et al. [43] conducted a large-scale empirical study of 16,221 Java projects on Github and found that licensing changes are predominantly toward or between permissive licenses, which is consistent with what we found in the license change. German et al. [25] proposed a method to detect compatibility between licenses declared in packages and those declared in source code, and to audit Fedora-12 software packages. Instead, we focus in this article on compatibility between the licenses declared in the package and the licenses of their dependencies.

Compared to previous studies [34, 47, 49], we focus on data in package management platforms and empirically analyze license usage on a large scale across five package management platforms. Specifically, we statistically analyze SPDX licenses in package management platforms, and we believe that SPDX licenses can provide a more comprehensive picture of license usage. In terms of research perspective, we statistically analyze licenses in package management platforms from multiple perspectives rather than a specific aspect [31, 43, 47]. Our study covers license incompatibility, license changes, license evolution, license characteristics of core packages, license usage across platforms, and more. Our study provides a more comprehensive picture of license usage in package management platforms.

6 CONCLUSION

In this paper, we conduct a large-scale comparative study of licenses across five package management platforms, Maven, NPM, PyPI, RubyGems, and Cargo, based on metadata from these platforms. We statistically analyze the license usage of these package managers from multiple perspectives. Our study shows that there are still a lot of license irregularities in the package management platforms. We find that the permissive license is the most used in any platform, and that the MIT and Apache-2.0 licenses are particularly popular. We also find that proportion of license incompatibility in the core versioned package is higher than that in overall versioned packages for all four platforms except Cargo, and that the most common license incompatibility term is sublicense. We hope to collect information on licenses for more platforms in the future and analyze license usage over different time horizons to further validate the validity of our approach.

7 ACKNOWLEDGMENTS

This research/project is supported by the National Key Research and Development Program of China (No. 2021YFB2701102), the National Science Foundation of China (No.62372398, No.62141222, and U20A20173), the Fundamental Research Funds for the Central Universities (No. 226-2022-00064), and Ningbo Natural Science Foundation (No. 2023J292).

REFERENCES

[1] 1997. *CRAN.* https://cran.r-project.org/

[2] 2003. *PyPI.* https://pypi.org/

[3] 2004. *Maven.* https://maven.apache.org/

[4] 2004. *RubyGems.* https://rubygems.org/

[5] 2007. *Open Source Definition.* https://opensource.org/osd/

[6] 2010. *NPM.* https://www.npmjs.com/package/npm

[7] 2015. *Cargo.* https://crates.io/

[8] 2022. *Cargo data source.* https://static.crates.io/db-dump.tar.gz

[9] 2022. *Maven data source.* https://repo.maven.apache.org/maven2/.index/

[10] 2022. *npm data source.* https://replicate.npmjs.com/_all_docs

[11] 2022. *PyPI data source.* https://console.cloud.google.com/marketplace/product/gcp-public-data-pypi/pypi?_ga=2.219857497.-1185994749.1670227125&project=dataanalysis-368712

[12] 2022. *RubyGems data source.* https://rubygems.org/pages/data

[13] 2023. *mvnrepository.* https://mvnrepository.com/

[14] 2023. *Replication Package.* https://figshare.com/s/5f35cac93da06567b1ca

[15] 2023. *ScanCode.* https://github.com/nexB/scancode-toolkit

[16] 2023. *tldrlegal.* https://www.tldrlegal.com/

[17] Alfred V Aho and Margaret J Corasick. 1975. Efficient string matching: an aid to bibliographic search. *Commun. ACM* 18, 6 (1975), 333–340.

[18] Daniel A Almeida, Gail C Murphy, Greg Wilson, and Michael Hoye. 2019. Investigating whether and how software developers understand open source software licensing. *Empirical Software Engineering* 24 (2019), 211–239.

[19] Barry W. Boehm. 1987. Improving software productivity. *Computer* 20, 09 (1987), 43–57.

[20] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* 30, 1-7 (1998), 107–117.

[21] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[22] Massimiliano Di Penta, Daniel M German, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2010. An exploratory study of the evolution of software licensing. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1.* 145–154.

[23] Karl Fogel. 2005. *Producing open source software: How to run a successful free software project.* " O'Reilly Media, Inc.".

[24] 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* 14 (2012), 143–154.

[25] Daniel M German, Massimiliano Di Penta, and Julius Davies. 2010. Understanding and auditing the licensing of open source software distributions. In *2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 84–93.

[26] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21 (2016), 2035–2071.

[27] Georgia M Kapitsaki and Georgia Charalambous. 2019. Modeling and recommending open source licenses with findOSSLicense. *IEEE Transactions on Software Engineering* 47, 5 (2019), 919–935.

[28] Jingyue Li, Reidar Conradi, Christian Bunse, Marco Torchiano, Odd Petter N Slyngstad, and Maurizio Morisio. 2009. Development with off-the-shelf components: 10 facts. *IEEE software* 26, 2 (2009), 80–87.

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

[30] Ilyas Saïd Makari, Ahmed Zerouali, and Coen De Roover. 2022. Prevalence and Evolution of License Violations in npm and RubyGems Dependency Networks. In *Reuse and Software Quality: 20th International Conference on Software and Systems Reuse, ICSR 2022, Montpellier, France, June 15–17, 2022, Proceedings*. Springer, 85–100.

[31] Yuki Manabe, Yasuhiro Hayase, and Katuro Inoue. 2010. Evolutional analysis of licenses in FOSS. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*. 83–87.

[32] M Douglas McIlroy, J Buxton, Peter Naur, and Brian Randell. 1968. Mass-produced software components. In *Proceedings of the 1st international conference on software engineering, Garmisch Pattenkirchen, Germany*. 88–98.

[33] Rômulo Meloca, Gustavo Pinto, Leonardo Baiser, Marco Mattos, Ivanilton Polato, Igor Scaliante Wiese, and Daniel M German. 2018. Understanding the usage, impact, and adoption of non-osi approved licenses. In *Proceedings of the 15th International Conference on Mining Software Repositories*. 270–280.

[34] Joao Pedro Moraes, Ivanilton Polato, Igor Wiese, Filipe Saraiva, and Gustavo Pinto. 2021. From one to hundreds: multi-licensing in the JavaScript ecosystem. *Empirical Software Engineering* 26 (2021), 1–29.

[35] 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.

[36] Shi Qiu, Daniel M German, and Katsuro Inoue. 2021. Empirical study on dependency-related license violation in the javascript package ecosystem. *Journal of Information Processing* 29 (2021), 296–304.

[37] Carlos Denner dos Santos. 2017. Changes in free and open source software licenses: managerial interventions and variations on project attractiveness. *Journal of Internet Services and Applications* 8 (2017), 1–12.

[38] SPDX. 2023. *SPDX License List*. https://spdx.org/licenses/

[39] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239* (2022).

[40] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[41] Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. 2017. License usage and changes: a large-scale study on github. *Empirical Software Engineering* 22 (2017), 1537–1577.

[42] Christopher Vendome, Daniel M German, Massimiliano Di Penta, Gabriele Bavota, Mario Linares-Vásquez, and Denys Poshyvanyk. 2018. To distribute or not to distribute? why licensing bugs matter. In *Proceedings of the 40th International Conference on Software Engineering*. 268–279.

[43] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. 2015. License usage and changes: a large-scale study of java projects on github. In *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 218–228.

[44] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. 2017. Machine learning-based detection of open source license exceptions. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 118–129.

[45] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M German, and Denys Poshyvanyk. 2015. When and why developers adopt and change software licenses. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 31–40.

[46] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. 2016. A look at the dynamics of the JavaScript package ecosystem. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 351–361.

[47] Thomas Wolter, Ann Barcomb, Dirk Riehle, and Nikolay Harutyunyan. [n. d.]. Open Source License Inconsistencies on GitHub. *ACM Transactions on Software Engineering and Methodology* ([n. d.]).

[48] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M German, and Katsuro Inoue. 2015. A method to detect license inconsistencies in large-scale open source projects. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 324–333.

[49] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M German, and Katsuro Inoue. 2017. Analysis of license inconsistency in large collections of open source projects. *Empirical Software Engineering* 22 (2017), 1194–1222.

[50] Sihan Xu, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu, and Hua Ji. 2023. LiDetector: License Incompatibility Detection for Open Source Software. *ACM Transactions on Software Engineering and Methodology* 32, 1 (2023), 1–28.

[51] Weiwei Xu, Hao He, Kai Gao, and Minghui Zhou. 2023. Understanding and Remediating Open-Source License Incompatibilities in the PyPI Ecosystem. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 178–190.