Work in Progress Paper: Detecting Method Level License Conflicts in the Worldwide Software Ecosystem

Aminul Didar Islam[1,2], Krishna Kaipa[2] and Slinger Jansen[1,2]

[1]LUT University, Yliopistonkatu 34, 53850 Lappeenranta, Finland
[2]Utrecht University, Utrecht, The Netherlands

Abstract

Building software relies on utilizing existing code and libraries, many of which are open source and come with an obligation to comply with its license. Non-compliance is considered unethical and opens up the possibility of a legal battle. Current tools to find violations extract license information from text assuming that the licenses are declared for borrowed code. This only works half the time as found by previous studies. Code level license extraction and violation checking is a definite way of determining license compliance for borrowed code, regardless of declaration. We demonstrate this using SearchSECO. We analyzed 3,500 repositories from top software companies to analyze the extent of violations and found around 32,000 violations in total. We then dissect these metrics to understand why these violations are occurring and how they can be mitigated.

Keywords

License conflicts, License management, Risk assessment

1. Introduction

The open-source movement has transformed the software industry, creating a vast and interconnected ecosystem where developers can freely share, reuse, and build upon each other's work. This worldwide ecosystem is composed of millions of software projects hosted on platforms like GitHub, GitLab, and Bitbucket, allowing developers to accelerate innovation by repurposing existing code modules, libraries, and even individual methods [1]. However, in the modern software development industry, code re-usability has emerged as a practice [2, 3]. The widespread reuse of code across this ecosystem introduces significant software licensing challenges. Each open-source component is typically governed by its own license, which outlines the terms under which the code can be used, modified, and redistributed. While licenses such as MIT, Apache, GPL, and others enable open collaboration, they also impose specific conditions that can lead to conflicts when incompatible licenses are combined in a single project [4]. Developers frequently utilize existing code and libraries to accelerate the software development process [2]. A significant proportion of these resources is open source, publicly accessible, and freely usable. However, the use of software code is governed by licenses, which impose certain obligations on users to ensure appropriate attribution, distribution of modifications, and preservation of user freedoms.

Regrettably, many software companies have been found to leverage open source software in their products without fully complying with the licensing terms. Such non-compliance is not only unethical but also potentially litigious, leading to serious legal consequences. There are many examples of these violations that have led to lawsuits which include companies like VMware, Vizio among many other companies in the electronics and software industry [5]. Despite this, the current tools for detecting license violations are largely dependent on extracting license information from the textual declarations accompanying the borrowed code. As demonstrated by previous studies, this approach is only successful about half the time, given that licenses are not always explicitly declared [6].

In response to this situation, our research had a twofold goal. First, to expand the set of tools available for identifying license violations by proposing a novel methodology for code-level license extraction

BENEVOL24: The 23rd Belgium-Netherlands Software Evolution Workshop, November 21-22, Namur, Belgium
aminul.islam@lut.fi (A. Didar Islam); kaipakrishna380@gmail.com (K. Kaipa); slinger.jansen@uu.nl (S. Jansen)
0009-0005-4792-4256 (A. Didar Islam); 0000-0003-3752-2868 (S. Jansen)

and violation checking. This method offers a more reliable way of determining license compliance for borrowed code, regardless of its declaration status. Second, we aimed to analyze the extent of these violations in the industry, focusing on major software companies.

To achieve these objectives, we leveraged the capabilities of SearchSECO, an open-source ecosystem mining project at Utrecht University. We analyzed 3,503 repositories from top software companies and found around 36,884 license violations. Our findings shed light on the extent of these violations, their validty, and severity, offering valuable insights for developers, organizations, and legal practitioners in the field of software licensing.

2. Background

2.1. Open Source Software Licensing

Open source software (OSS) is built on the philosophy of free sharing and collaboration [7]. OSS licenses are the legal mechanisms that protect this philosophy by stipulating the terms of use, modification, and distribution of the software. The terms may vary widely across different licenses, ranging from permissive licenses like the MIT License, which only requires attribution and preserves the freedom to use, modify, and distribute the software as desired, to copyleft licenses like the GNU General Public License (GPL), which requires any derivative works to be distributed under the same license. OSS licenses can be broadly categorized into four types, each with its unique stipulations and requirements:

1. **Permissive Licenses**: Also known as "liberal" licenses, permissive licenses are the least restrictive among OSS licenses. They allow users to use, modify, distribute, and even sublicense the software with very few obligations. These licenses generally only require users to retain the original copyright notice and disclaimer in the source code. Examples include the MIT License and the Apache License 2.0.
2. **Weak Copyleft Licenses**: These licenses are a balance between permissive and strong copyleft licenses. They allow users to link to libraries or include code in larger works without requiring the entire work to be open sourced. However, any changes to the original open source component must be made available under the same license. An example is the Lesser General Public License (LGPL).
3. **Strong Copyleft Licenses**: These licenses, also known as "reciprocal" licenses, are more restrictive. They allow for free use, modification, and distribution of the software, but require that any derivative work or distribution of the software (including modifications) also be licensed under the same terms, thereby preserving the "openness" of the software. The most well-known strong copyleft license is the GNU General Public License (GPL).
4. **Other**: This category encompasses licenses that do not fit into the above categories or are less commonly used. Examples include proprietary-like open source licenses, which allow for free use and modification of the software but place restrictions on the distribution of modified works. Other licenses, like the Creative Commons licenses, are designed for works other than software, but are sometimes used in software-related contexts, particularly for documentation or other non-code assets.

In summary, OSS licenses ensure the sharing and collaboration of open source software by defining how it can be used, modified, and distributed among communities. These licenses range from permissive, with minimal restrictions, to strong copyleft, which requires derivative works to maintain the same open source terms and conditions.

2.2. Key Reasons for Verifying Open-Source Licenses

1. Legal and Financial Risks: Research emphasizes that non-compliance with open-source licenses can lead to legal consequences, including fines, loss of intellectual property rights, and even

litigation. Studies show that many companies face compliance challenges due to the diversity of licenses and their varying requirements [8].

2. Intellectual Property Management: Open-source license compliance ensures that organizations can maintain control over their proprietary code by managing how open-source components interact with it. This is crucial, especially in products where license conflicts can inadvertently force companies to release proprietary code if they use copyleft licenses like GPL. This aspect is critical for long-term IP strategy Different components might be under different licenses and merging those under one license can conflict [9].
3. Security: Licensing checks also enable better management of software security, as they facilitate a clearer inventory of all open-source components and their vulnerabilities. This approach is becoming a best practice in software supply chains, where security risks can arise from poorly monitored or conflicting licenses [10]. For example, pushing a poor code by someone to Linux can make the OS performance slower. In addition, how we distribute open-source software is also defined by the license.
4. Community involvement and Ethical Responsibility: Complying with open-source licenses ensures fair attribution and respects contributors' work. This respect strengthens open-source ecosystems and encourages continued community involvement, which benefits both developers and end-users [8]. In addition, open-source code can't be mixed with non-open-source code.

2.3. License Scanners

Open source license compliance is often ensured through the use of automated license scanners, tools that are engineered to discern the licensing terms of software components. For example, two such prominent tools are Nomos[1] and ScanCode [11] (more examples are provided in the Related Work Section).

Nomos, an integral part of the FOSSology project—an open-source license compliance software system—operates by scanning files within a software package to detect license-related statements. Its functionality is rooted in heuristic pattern matching, employing a database of license templates to equate the text in the scanned files with known licenses. However, the efficiency of Nomos is constrained by its reliance on explicit license declarations in the code. If the license isn't declared, or if the declaration deviates from a format Nomos recognizes, it may not succeed in correctly identifying the license.

On the contrary, ScanCode serves as a comprehensive toolkit for scanning code. Its capabilities extend to the extraction and identification of licenses, copyrights, and other legal notices from code files. Although ScanCode shares Nomos' reliance on pattern matching against a database of known licenses, it employs a more intricate matching approach, enabling the identification of more granular licensing terms within a file. Despite its sophistication, ScanCode's effectiveness is similarly restricted by the necessity for explicit license declarations in the code files. In their absence, ScanCode might struggle to accurately discern the licensing terms.

These limitations inherent to license scanners underscore the complexities involved in ensuring license compliance in open source software. They highlight the necessity for advanced methods like the one proposed in this research - a code-level license extraction and violation checking approach that can identify license terms irrespective of explicit license declarations.

2.4. SearchSECO

SearchSECO[2] functions as a hash-based index for code fragments and enables the exploration of source code at the method level across the global software ecosystem [12]. By extracting and indexing data at this granular level, SearchSECO makes individual code methods and their associated metadata searchable, providing a detailed view of software repositories that is often missing from broader analyses. Data gathering for SearchSECO follows a systematic approach:

[1] https://www.fossology.org/features

[2] https://github.com/SecureSECO/searchSECO-miner, documentation at https://docs.secureseco.org/searchseco-integration.

Figure 1: License violation detection flow

1. Repository Gathering: The initial step involves SearchSECO identifying and collecting repositories based on their popularity, gauged by the number of stars on platforms like GitHub. These repositories are then arranged for parsing, prioritized by the number of files they contain.
2. Parsing and Extraction: Each repository in the queue undergoes parsing, where methods from each file are identified and extracted. This extraction is designed to be language-independent, capturing methods irrespective of the programming language employed.
3. Hashing and Storage: Post extraction, the methods are hashed using the MD5 algorithm, a common cryptographic hash function that outputs a unique 128-bit hash for each method. This hash, along with other metadata like version number and license information, is stored in the SearchSECO database. This efficient hashing enables streamlined storage and retrieval of methods while ensuring their uniqueness.
4. Querying: Users can interrogate the database with a specific method to find code clones [13]. If the method exists in the database, the system returns matching methods along with their associated metadata.

The functionality of SearchSECO, in identifying and storing code fragments and their related metadata, including license information, establishes it as a powerful tool for code-level license extraction and violation detection, as exemplified in this research. What makes SearchSECO unique, is that it can detect code clones up to the level where variable names are changed and values and operators may have changed. While this could lead to false positives, we can also find obvious code plagiarism that is not typically detected through other methods, such as file comparisons.

By surpassing the limitations of conventional license scanners, SearchSECO introduces a promising avenue for ensuring license compliance in open source software development.

3. Related work

OSS can be distributed under multiple licenses, when the same project contains two different licenses, it is called inconsistency. However, inconsistency doesn't necessarily comply with a license conflict [14]. Copyleft licenses such as the GPL family of licenses provide a restriction that any code changes should be publicly available under the same license on the other hand permissively licensed code allows to creation of closed proprietary software. License compatibility poses a potential challenge, as not all licenses are mutually compatible [15, 14]. In addition, one part of the software may be dependent on another part of the software and according to that license behave differently [14].

Recent studies have shed light on licensing conflicts in the open-source software ecosystem. One of the most notable is the work of Wolter et al., July 2023, which delves into the GitHub ecosystem, aiming to identify non-compliance between declared licenses and in-code licenses [6]. Their findings indicate a significant mismatch: around half of the repositories failed to declare all in-code licenses. Of these, nearly ten percent presented a mismatch between permissive and copyleft licenses. Wolter et al. relied on license scanners like Nomos and ScanCode which, while powerful tools, overlook libraries or code clones without explicit license declarations. Their work highlights the need for tools that analyze licenses directly from code, motivating our approach in this research.

The study conducted by Feng et al. focused on license conflicts in binary files at a large scale [16]. The researchers attempted to map the components in an application binary to their contributing repositories, including libraries and imported code. This was accomplished using code features and file attributes. The study's primary focus was on binary software and its mapping to open software repositories. By identifying potential license conflicts in binary files, the researchers aimed to provide insights into the licensing practices of software organizations and identify potential areas of risk. One limitation of the study is that it only focused on software with binary releases and did not consider in-code licenses for any software repository. This means that the study may have overlooked potential license conflicts that arise from code borrowing or other forms of in-code use.

Timo et al. (2009), introduced one interesting project named ASLA followed a reverse engineering approach and did a dependency analysis identifying the dependencies between objects (source files, compiled objects, libraries, etc.) [17]. The paper asserts that ASLA is the only license analysis tool currently available that offers comprehensive information on build process outputs and their dependencies. While ASLA is a powerful tool, it is limited by its compatibility with only Java version 1.6.0_03 and Linux-based operating systems. In addition, it doesn't support language that cannot be compiled using GCC and (or) does not produce the required dependency information files.

Gianluca De Bonis (2022), studied and worked on four existing popular license compliance code scanning tools and reported details of usability for developers [18]. FOSSology is an open source project introduced in 2007 and later chronologically they released more versions including license scanners and updating over time. For license scanning, they are also incorporating open source license scanners such as Ninka [19]. Ninka analyzes sentences within source code comments, it can recognize over 120 different licenses [20]. However, Fossology identifies the licenses of open-source software but does not provide detailed information, such as license versions, as it lacks a dedicated database [21]. One of the popular license checkers Scancode runs on multiple platforms from the command line but the runtime is very slow. Another tool is License Finder is a scanning tool that integrates with the project's package manager to detect the licenses of both the project and its dependencies. Most simplest license scanner is Licensee with limited but effective options [18].

A study Sihan Xu et al. (2023) [22] introduces LiDetector, an automated tool designed to assess license texts in open-source software, identifying rights and obligations to detect license incompatibility. LiDetector works with both standard and custom licenses, improving upon previous methods by achieving a 10.06% false positive rate and a 2.56% false negative rate in tests. In a large-scale analysis of 1,846 GitHub projects, LiDetector found that 72.91% had license incompatibility issues, often involving widely used licenses like MIT and Apache. This highlights the need for greater awareness among developers and companies.

Sihan Xu et al. (2023) published another study [23] presents LiResolver, a tool designed to automatically interpret software licenses, pinpoint license incompatibility issues, and provide actionable solutions for licensors. LiResolver is fine-grained, scalable, and flexible, with experiments showing it achieves a 4.09% false positive rate and 0.02% false negative rate in identifying incompatibility issues. Remarkably, it resolved 62.61% of incompatibility issues in a test set of 230 real-world projects.

Golubev et al.'s research centered on license conflicts resulting from code borrowing and imported libraries, but only within Java projects on GitHub [24]. Their comprehensive evaluation of licensing conflict metrics revealed concerning levels of potential code borrowing and license violation. However, the language-specific focus of the study underscores the need for a language-agnostic methodology, which our research strives to provide. Massimiliano et al. also presented another language-specific solution for Java programming language: an automated method for identifying the license of Jar archives, which integrates a code search engine with automated classification of licenses found in the textual files within the Jar [25].

Finally, software distributed in binary form often includes third-party packages without adhering to their licenses, leading to frequent, unintentional license violations. A common example is the inclusion of the Linux kernel in device firmware without fulfilling the requirements of the GNU General Public License. To address this, the Binary Analysis Tool (BAT) was developed to detect code clones in binaries, helping to identify potential license breaches. BAT analyzes binaries like firmware images by comparing

them to known sources and binary repositories using techniques such as string literal scanning, data compression similarity, and binary delta computation [26].

These studies collectively highlight the growing concern around license conflicts in software development, underscoring the need for tools and methodologies like ours, which aim to provide a more comprehensive and accurate picture of license compliance across the software ecosystem.

4. Research Method

The research method employed involves the development of the license violation detection module leveraging SearchSECO's capabilities. This is then utilised to perform an analysis on the repositories from software companies selected using the detailed criteria.

	Apache 2.0	MIT	GPL-2.0-only	BSD-2-Clause	GPL-3.0-only
Apache 2.0	Same	Yes	No	Yes	No
MIT	Yes	Same	No	Yes	No
GPL-2.0-only	No	Yes	Same	Yes	No
BSD-2-Clause	Yes	Yes	No	Same	No
GPL-3.0-only	Yes	Yes	No	Yes	Same

Table 1
Shows a subsection of the license compatibility matrix

The license violation module is built on top of SearchSECO and into the client program called the controller. The controller is capable of operating autonomously in which it mines the various software ecosystems like GitHub, GitLab etc. and the second mode of operation is to check a project for license violations and security vulnerabilities. The controller can be accessed via a frontend deployed by us at Utrecht University or using the published docker container, but it can also be built by yourself using code from GitHub). The code is executable on linux natively and on any operating system using Docker. The code is written in C++. There are several commands that can be run using the controller but for the purposes of this study, we will be using the "check" command which checks a given repository against the SearchSECO database. It takes the repository to be checked for license violations as input. The controller has four modules, spider, parser, crawler, and database API connection module.

The process of detecting license violations operates on a method-by-method basis, with each repository serving as an input for the process. This process can be broken down into several distinct steps and is shown in figure 1:

1. **Repository Parsing and Hash Generation**: The first step in the process involves parsing all the files in the provided repository. The repository is downloaded with the spider and handed to the parser. The methods within these files are identified, and hashes are generated for each of these methods.
2. **Database Query**: The generated hashes are then queried against the SearchSECO database to identify any matches. The precision of the hash function ensures that the matches correspond to specific methods in the repositories even if the variable and function names are changed. Code clones are detected which are functionally the same.
3. **License Determination**: The licenses associated with the matched methods are identified based on the license of the parent repository where the method is found. This provides a method-specific license, which can be used for a detailed license compatibility check.
4. **License Compatibility Check**: To determine whether the licenses are compatible, we refer to a license compatibility matrix. This matrix, provided by the Open Source Automation Development Lab (OSADL), lists the compatibility of different open-source licenses with each other[3]. By comparing the license of the matched method with that of the repository under investigation, we

[3]https://www.osadl.org/

can determine if there is a licensing conflict. The compatibility matrix shown in table 1 shows a portion of the compatibility matrix that we have used in the development of this software. To determine the compatibility of lets say, usage of GPL 2.0 code under Apache 2.0 license, we find the Apache 2.0 row and check the GPL 2.0 column. In this case, it is not possible so we get a "No".

5. **Report Generation**: Once the license compatibility checking process is complete for all methods in the repository, a comprehensive report is generated. This report includes the total number of detected license violations, as well as detailed information about each matched method, including its associated license, the repository it originates from, and any license compatibility issues.

The development of our license violation detection technique offers utility for practitioners and software producing organization. By facilitating method-level license checks, it enables a more thorough and precise detection of potential license violations. This, in turn, can help practitioners ensure their code's compliance with open-source licenses and help companies mitigate the risk of legal and ethical implications arising from unintentional license violations. This tool serves not only as a research instrument but also as a practical solution for fostering adherence to open-source licensing norms in the software development industry.

5. Results

In order to investigate the extent of license violations within software companies, we first established selection criteria for these organizations and their corresponding repositories. Our criteria centered on the total number of stars accumulated across all of an organization's public repositories on GitHub. While we acknowledge the potential limitations of this metric, it does provide a list of repositories from a wide array of software companies that not only have a substantial open source presence but also enjoy popularity within the open source community.

Based on this criterion, we identified the top five companies with the highest starred repositories:

Organisation	Number of repositories
Microsoft	5,561
Google	2,504
Apache	2,465
Alibaba	388
Facebook	126

These companies collectively maintain 11,044 repositories. At the time of this report, we have analyzed 3,910 of these repositories. In subsequent sections, we provide a detailed account of each metric and observation from our analysis. We made the results open to the community, the first version of the results can be found at [27].

5.1. Total Candidate License Violations

One of our significant findings was the identification of a total of 32,011 candidate license violations across all analyzed repositories, suggesting a widespread occurrence of violations. It should be noted that this figure comprises violations both from repositories that have sourced code from the repositories under examination, as well as violations within the inspected repositories themselves.

Interestingly, a large proportion of the analyzed repositories employed Apache and MIT licenses, which tend to pose fewer issues when their code is reused. Problems primarily arise when these licenses incorporate more restrictive code. Consequently, the high number of violations discovered in our analysis suggests a significant number of these infringements might indeed represent legitimate areas of concern.

Organisation	Number of checked repositories	Total number of candidate violations	Number of candidate violations per repository
Microsoft	1,521	4,525	3.0
Google	1,657	27,176	16.4
Apache	102	2,964	29.1
Alibaba	113	2,185	19.3
Facebook	110	34	0.3

Table 2
Shows total number of candidate violations and number of candidate violations per repository. Please note that our approach is method-based, so when a project includes many methods from another project, it will identify a significant number of violations.

5.2. Total License Violations Found by Company

In our analysis of license violations, we observed that the highest number of violations were found within the repositories of Google, followed closely by Microsoft. This observation could potentially be explained by the sheer volume of repositories maintained by these two organizations. Furthermore, we must be aware that a candidate violation is not necessarily a violation, as further research is needed here. Also, violations could overlap; two candidate violations could stem from the same 'real' license violation, as there are multiple methods per file that are part of the violation. More research is needed here, and we consider that future work.

In order to obtain a more nuanced understanding of the extent of license violations, we also calculated the number of candidate violations per repository for each organization. This metric serves as a more accurate reflection of the prevalence of license violations, as it adjusts for the varying numbers of repositories across different organizations.

Table 2 presents the number of total candidate violations alongside the calculated violations per repository for each organization. It is important to note that a higher number of violations per repository may indicate a more prevalent issue of non-compliance within the organization's software development practices. These findings highlight the importance of license compliance and the potential utility of our license violation detection module in identifying and addressing such issues. That said, this paper is a work in progress, and the actual violations still need to be identified before a full analysis can be done.

6. Discussion

6.1. Analysis and Repercussions

Our findings reveal a significant number of license violations across some of the most influential software companies in the industry. This prevalence of license violations may be due to a number of factors, including the complexity and diversity of open source licensing terms, inadequate understanding of these terms among developers, and the lack of robust tools for license compliance checking.

The repercussions of these violations can be severe, ranging from damaged reputation to potential legal consequences. In addition, such violations undermine the spirit of the open source community, which is based on mutual respect for each other's work and the terms under which it is shared.

To mitigate these issues, there are a few strategies companies could adopt. Firstly, raising awareness and improving education about open source licenses among developers can help prevent inadvertent violations. Secondly, integrating license compliance checks as part of the development and deployment pipeline can ensure that violations are detected and addressed promptly. Lastly, using tools like SearchSECO can provide a more granular and reliable method of license checking, thereby reducing the risk of violations.

6.2. Threats to Validity

While our study offers valuable insights into license violations in the software industry, there are several threats to its validity.

Firstly, our selection criteria of companies is based on the number of stars on their GitHub repositories. While this metric offers a proxy for the companies' influence and popularity in the open source community, it does not account for the entirety of their work, especially the code that isn't publicly available or not hosted on GitHub.

Secondly, we relied on SearchSECO's database for detecting license violations. Although this tool provides a more granular approach to license checking, its effectiveness is limited by the completeness and accuracy of its code fragment database. Also, we believe that an in depth review at the code snippets would show duplicated violations of the same code. This has to be investigated in further detail. Code not included or incorrectly indexed in the database could lead to false negatives.

Lastly, our analysis assumes that the license of a code fragment is determined by the license of the repository it is found in. This approach may not capture complex licensing scenarios where different parts of a repository may be under different licenses.

Despite these limitations, we believe that our study provides a crucial first step in understanding the prevalence of license violations in the software industry and highlights the need for more robust methods for ensuring license compliance.

7. Future Work

Given the extensive number of license violations found in our study, it is clear that further investigation into the root causes of these occurrences is needed. An in-depth analysis of individual violations could provide valuable insights into the circumstances and factors contributing to these infractions. Understanding the 'why' behind these violations could not only aid in devising more effective prevention and mitigation strategies, but also promote a more responsible and informed open source community.

Conducting surveys or interviews with developers from the examined organizations could shed light on their awareness and understanding of open source licenses and violations. This qualitative approach could help us discern whether the license violations are occurring due to negligence, lack of knowledge, or potentially, malicious intent.

Our analysis also highlights the need for more sophisticated reporting systems. The number of violations detected by our study might be inflated by the duplicate detection of the same code in multiple repositories. In reality, the code may originate from a single repository and is merely being borrowed by others. Thus, a system capable of detecting and reporting the highest level of code copying would provide a more accurate picture of license violations.

Furthermore, extra study is needed in the use of so-called license evasion techniques [28], which avoid problems with reuse. One such technique is introducing a client-server architecture, which in some cases avoids the formal license conflict. With tools from the FASTEN project [29], which does code analysis to see how dependencies are used, we can evaluate the extent of a license conflict and in the future propose fixes.

Additionally, future research could explore the development of more advanced license checking tools that can handle complex licensing scenarios and track the propagation of code across repositories. With the increasing dependence on open source software, ensuring license compliance is not only a legal necessity but also a matter of ethical responsibility.

Finally, it would be interesting to extend the scope of our study to other prominent software companies and open source projects. This would provide a more comprehensive understanding of the extent and impact of license violations in the software industry and help open source receive the attention and credit that it deserves.

8. Conclusion

This study highlights the critical importance of license compliance in software development, especially given the widespread use of open-source code. Current tools, which rely primarily on extracting license information from textual declarations, often fail to detect violations when licenses are not explicitly stated—a limitation that makes compliance monitoring unreliable. To address this, we introduced a code-level approach to license extraction and compliance verification, demonstrated through SearchSECO, which significantly improves the detection accuracy of license violations.

Our analysis of 3,503 repositories from leading software companies uncovered nearly 37,000 license violations, underscoring the pervasiveness of non-compliance within the industry. This extensive count not only reveals the potential ethical and legal risks organizations face but also emphasizes the need for a more robust and reliable method of tracking license adherence at the code level. By dissecting the patterns and underlying causes of these violations, we provide insights that may help mitigate future issues, offering value to developers, companies, and legal professionals.

In conclusion, this research underscores the urgency for adopting more reliable compliance tools and practices. Code-level license checking, as demonstrated, offers a promising path forward to ensure that the use of open-source code remains ethical, lawful, and sustainable in the long term. Future studies could further refine these methods and explore additional strategies to address the challenges of open-source license management.

References

[1] O. Carlsson, H. Sjölinder, Open Source Software Licenses Impact on Businesses, 2023. URL: https://www.diva-portal.org/smash/get/diva2:1767854/FULLTEXT02.pdf, dissertation.

[2] Y. Huang, F. Xu, H. Zhou, X. Chen, X. Zhou, T. Wang, Towards Exploring the Code Reuse from Stack Overflow during Software Development, in: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 548–559. doi:10.1145/3524610.3527923.

[3] S. Haefliger, G. Von Krogh, S. Spaeth, Code Reuse in Open Source Software, Management science 54 (2008) 180–193.

[4] A. Onetti, S. Verma, Open Source Licensing and Business Models, IUP Journal of Knowledge Management 7 (2009) 68–94. URL: https://www.proquest.com/scholarly-journals/open-source-licensing-business-models/docview/199279423/se-2?accountid=27468, aBI/INFORM Collection; SciTech Premium Collection.

[5] G. Gangadharan, S. De Paoli, V. D'Andrea, M. Weiss, License Compliance Issues in Free and Open Source Software., MCIS 2008 Proceedings (2008) 2. URL: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1010&context=mcis2008.

[6] T. Wolter, A. Barcomb, D. Riehle, N. Harutyunyan, Open Source License Inconsistencies on GitHub, ACM Trans. Softw. Eng. Methodol. (2022). doi:10.1145/3571852.

[7] M. Papoutsoglou, G. M. Kapitsaki, D. German, L. Angelis, An Analysis of Open Source Software Licensing Questions in Stack Exchange Sites, Journal of Systems and Software 183 (2022) 111113. doi:https://doi.org/10.1016/j.jss.2021.111113.

[8] X. Cui, J. Wu, Y. Wu, X. Wang, T. Luo, S. Qu, X. Ling, M. Yang, An Empirical Study of License Conflict in Free and Open Source Software, in: 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), IEEE, 2023, pp. 495–505. doi:10.1109/ICSE-SEIP58684.2023.00050.

[9] A. Mathur, H. Choudhary, P. Vashist, W. Thies, S. Thilagam, An Empirical Study of License Violations in Open Source Projects, in: 2012 35th annual IEEE software engineering workshop, IEEE, 2012, pp. 168–176.

[10] P. Ombredanne, Free and Open Source Software License Compliance: Tools for Software Composition Analysis, Computer 53 (2020) 105–109. doi:10.1109/MC.2020.3011082.

[11] How Does ScanCode Detect Licenses?, 2023. URL: https://scancode-toolkit.readthedocs.io/en/latest/reference/overview.html, [www document], [Accessed on 12.09.2024].

[12] S. Jansen, S. Farshidi, G. Gousios, J. Visser, T. van der Storm, M. Bruntink, SearchSECO: A Worldwide Index of the Open Source Software Ecosystem, in: BENEVOL, 2020. URL: https://pure.rug.nl/ws/portalfiles/portal/192817754/paper3.pdf.

[13] N. Saini, S. Singh, Suman, Code Clones: Detection and Management, Procedia Computer Science 132 (2018) 718–727. doi:https://doi.org/10.1016/j.procs.2018.05.080, International Conference on Computational Intelligence and Data Science.

[14] T. Tuunanen, J. Koskinen, T. Karkkainen, Asla: Reverse Engineering Approach for Software License Information Retrieval, in: Conference on Software Maintenance and Reengineering (CSMR'06), 2006, pp. 4 pp.–294. doi:10.1109/CSMR.2006.10.

[15] M. Mustonen, Copyleft—the Economics of Linux and Other Open Source Software, Information Economics and Policy 15 (2003) 99–121. doi:https://doi.org/10.1016/S0167-6245(02)00090-2.

[16] M. Feng, W. Mao, Z. Yuan, Y. Xiao, G. Ban, W. Wang, S. Wang, Q. Tang, J. Xu, H. Su, B. Liu, W. Huo, Open-Source License Violations of Binary Software at Large Scale, in: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019, pp. 564–568. doi:10.1109/SANER.2019.8667977.

[17] T. Tuunanen, J. Koskinen, T. Kärkkäinen, Automated Software License Analysis, Automated Software Engineering 16 (2009) 455–490. doi:10.1007/s10515-009-0054-z.

[18] G. De Bonis, License Compliance Code Scanning Tools For Developers, Technical Report, 2022.

[19] M. C. Jaeger, O. Fendt, R. Gobeille, M. Huber, J. Najjar, K. Stewart, S. Weber, A. Wurl, The Fossology Project: 10 Years of License Scanning, International Free and Open Source Software Law Review (IFOSS L. Rev.) 9 (2017) 9. doi:10.5033/ifosslr.v9i1.123.

[20] S. Van Der Burg, E. Dolstra, S. McIntosh, J. Davies, D. M. German, A. Hemel, Tracing Software Build Processes to Uncover License Compliance Inconsistencies, in: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 731–742. doi:https://doi.org/10.1145/2642937.2643013.

[21] KimKi-Hwan, YoonSeong-Cheol, KimSu-Hyun, LeeIm-Yeong, A Study on Open Source Version and License Detection Tool, The Transactions of the Korea Information Processing Society 13 (2024) 299–310. doi:https://doi.org/10.3745/TKIPS.2024.13.7.299.

[22] S. Xu, Y. Gao, L. Fan, Z. Liu, Y. Liu, H. Ji, LiDetector: License Incompatibility Detection for Open Source Software 32 (2023). URL: https://doi.org/10.1145/3518994. doi:10.1145/3518994.

[23] S. Xu, Y. Gao, L. Fan, L. Li, X. Cai, Z. Liu, LiResolver: License Incompatibility Resolution for Open Source Software, ISSTA 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 652–663. URL: https://doi.org/10.1145/3597926.3598085. doi:10.1145/3597926.3598085.

[24] Y. Golubev, M. Eliseeva, N. Povarov, T. Bryksin, A Study of Potential Code Borrowing and License Violations in Java Projects on GitHub, in: Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 54–64. doi:10.1145/3379597.3387455.

[25] M. Di Penta, D. M. German, G. Antoniol, Identifying Licensing of jar Archives Using a Code-search Approach, in: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), 2010, pp. 151–160. doi:10.1109/MSR.2010.5463282.

[26] A. Hemel, K. T. Kalleberg, R. Vermaas, E. Dolstra, Finding Software License Violations Through Binary Code Clone Detection, in: Proceedings of the 8th Working Conference on Mining Software Repositories, 2011, pp. 63–72. doi:https://doi.org/10.1145/1985441.19854.

[27] S. Jansen, A. D. Islam, SearchSECO License Conflicts Metadata, 2024. URL: https://doi.org/10.5281/zenodo.13753110, [www document], [Accessed on 12.09.2024].

[28] Sufatrio, D. J. Tan, T.-W. Chua, V. L. Thing, Securing Android: A Survey, Taxonomy, and Challenges, ACM Computing Surveys (CSUR) 47 (2015) 1–45. doi:https://doi.org/10.1145/2733306.

[29] J. Hejderup, M. Beller, K. Triantafyllou, G. Gousios, Präzi: From Package-based to Call-based Dependency Networks, Empirical Software Engineering 27 (2022) 102. URL: https://link.springer.

com/article/10.1007/s10664-021-10071-9.