Your "Notice" Is Missing: Detecting and Fixing Violations of Modification Terms in Open Source Licenses during Forking

Kaifeng Huang*
Tongji University
Shanghai, China

Yingfeng Xia†
Fudan University
Shanghai, China

Bihuan Chen‡†
Fudan University
Shanghai, China

Siyang He†
Fudan University
Shanghai, China

Huazheng Zeng†
Fudan University
Shanghai, China

Zhuotong Zhou†
Fudan University
Shanghai, China

Jin Guo†
Fudan University
Shanghai, China

Xin Peng†
Fudan University
Shanghai, China

Abstract

Open source software brings benefit to the software community but also introduces legal risks caused by license violations, which result in serious consequences such as lawsuits and financial losses. To mitigate legal risks, some approaches have been proposed to identify licenses, detect license incompatibilities and inconsistencies, and recommend licenses. As far as we know, however, there is no prior work to understand modification terms in open source licenses or to detect and fix violations of modification terms. To bridge this gap, we first empirically characterize modification terms in 48 open source licenses. These licenses all require certain forms of "notice" to describe the modifications made to the original work. Inspired by our study, we then design LiVo to automatically detect and fix violations of modification terms in open source licenses during forking. Our evaluation has shown the effectiveness and efficiency of LiVo. 18 pull requests for fixing modification term violations have received positive responses. 8 have been merged.

CCS Concepts

• **Software and its engineering → Software libraries and repositories**; *Software configuration management and version control systems*; **Software evolution**; *Software maintenance tools.*

Keywords

open source software, software license, license violation, software modification

ACM Reference Format:
Kaifeng Huang, Yingfeng Xia, Bihuan Chen, Siyang He, Huazheng Zeng, Zhuotong Zhou, Jin Guo, and Xin Peng. 2024. Your "Notice" Is Missing: Detecting and Fixing Violations of Modification Terms in Open Source Licenses during Forking. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3650212.3680339

*K. Huang is with the School of Software Engineering, Tongji University, China.
†Y. Xia, B. Chen, S. He, H. Zeng, Z. Zhou, J. Guo, X. Peng are with the School of Computer Science and Shanghai Key Laboratory of Data Science, FudanUniversity, China.
‡Bihuan Chen is the corresponding author.

ISSTA '24, September 16–20, 2024, Vienna, Austria

ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3680339

1 Introduction

Open source software has been widely embraced by developers, and the software community has been fostered from it tremendously in the past decade. Despite the positive impetus it brings, open source software carries inherent risks [66, 69], with one of the most significant being the legal risks caused by violations of licenses. Open source licenses regard open source software as intellectual property, and declare license terms that regulate the rights and obligations under which developers could conduct activities like reusing, distributing and modifying open source software.

There exists a large variety of open source licenses [36], ranging from permissive ones like MIT to restrictive ones like GPL. Failing to fulfill obligations in an open source license can cause license violation, also known as a license bug [60], which can hinder software development and prevent software from being released. One prominent example of license violations is the enforcement of copyleft terms in GPL. Copyleft requires that any modifications to the software must also be distributed under the same license as the original software, ensuring that the modified version remains open source like the original software. GPL has enforced compliance on more than 150 products, resulting in many lawsuits and causing financial losses to companies [31].

To mitigate legal risks, some approaches have been proposed to identify licenses, detect license violations, and recommend licenses. License identification is the first step towards automatic license analysis. It is realized by rule-based matching [16, 27, 28, 57] and machine learning [32, 42, 62]. License violations can be caused by various reasons. One line of work attempts to detect incompatibility among the licenses of a system and its declared dependencies or reused source code [41, 52, 70]. Another line of work tries to find violations of copyleft terms of GPL/AGPL licenses [19, 21, 31]. The other line of work tries to detect inconsistency between the licenses of two source files that are evolved from the same provenance by code reuse [25, 67, 68] and between the license of a package and the license of each file of the package [17, 24, 50]. To prevent license violations, license recommendation [40, 46] can be used to choose compliant licenses.

However, little attention has been paid to violations of modification terms (MTs) in open source licenses. MTs specify the conditions

under which users of open source software may modify the original software and the obligations they must fulfill. For example, the Apache-2.0 license specifies that *"You may reproduce and distribute copies of the Work ..., provided that you meet the following conditions: ... b. You must cause any modified files to carry prominent notices stating that You changed the files; ..."*. MTs distinguish liability, copyright, and benefit among shareholders when distributing derivative works. For example, the NGPL license dictates that *"If NetHack is modified by someone else and passed on, we want its recipients to know that what they have is not what we distributed"*. Besides, developers may also not be familiar with MTs, and thus discuss how to obligate MTs [1, 6, 10]. Unfortunately, to the best of our knowledge, there has been no effort to understand MTs in open source licenses or to detect and fix their violations automatically.

To fill this gap, we first conduct an empirical study on MTs of open source license approved by Open Source Initiative [36]. Of these 107 approved licenses, 48 licenses noticeably declare MTs, and require certain forms of "notice" to describe modifications made to the original work. To gain deeper understanding of MTs, we analyze license documents to identify similarities and differences in how MTs are implemented. We find that they differ in modification scope (i.e., the types of modified files), notice content (i.e., the description of modifications), and notice location (i.e., the location to put the notice). We summarize nine, six, and five categories of modification scope, notice content, and notice location, respectively, which are used to model MTs of the 48 studied licenses.

Inspired by our study, we then propose an automatic approach, named LiVo, to detect and fix violations of MTs during forking, where a fork repository is first created by copying a base repository and then modified separately. We specifically focus on the reuse and redistribution scenario of forking because i) it is widely used in open source software development, and ii) it is non-trivial to ensure compliance with MTs due to the complex modification history of the fork repository. Given a base and a fork repository, LiVo works in three steps. First, it identifies the obligating commits of the fork repository. These obligating commits modify the original files that are in the modification scope of the MT of the base repository. Second, it extracts change logs from potential notice locations in the fork repository. These changes logs contain potential notice for those obligating commits. Third, it matches change logs with commit logs based on the notice content of the MT to detect and fix MT violations. LiVo can be adopted by authors of the base repositories to detect MT violations, and by authors of the fork repositories to detect and fix MT violations such that potential legal risks can be avoided.

We evaluate the effectiveness and efficiency of LiVo using 178 pairs of base and fork repositories. 91 fork repositories violate MTs of the base repositories, while 87 fork repositories do not modify original files and hence fulfill MTs of the base repositories. LiVo achieves a precision of 0.82 and a recall of 0.80 in detecting and fixing MT violations. We submit 91 pull requests to report and fix MT violations. 18 of them have received positive responses from developers, and 8 have been merged. LiVo takes 229.8 seconds to detect and fix MT violations in each pair of base and fork repository.

In summary, our work makes the following contributions.

- We conducted the first empirical study on 48 open source licenses to characterize MTs.
- We proposed LiVo to detect and fix violations of MTs in open source licenses during forking.
- We evaluated to demonstrate the effectiveness and efficiency of LiVo. 18 pull requests for fixing MT violations have received positive responses, and 8 have been merged.

2 Motivation

The open source license was published decades ago. The first open source license can be traced back to the launch of the free software movement. In 1983, Richard Stallman launched the GNU Project to write a complete operating system free from constraints on use of its source code [5]. He used "free software" and founded the Free Software Foundation to promote the concept. The first version of the GNU General Public License [4] was published in 1989, which is known for its copyleft term. Specifically, the copyleft term from GPL mandates that users maintain their copies of the software as open source and extend identical rights to recipients, outlined in GPL's preamble [4]:

> *"For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights."*

However, although the early open source software was shipped together with its license, the legal rights of open source software developers remained unknown for a long time. In 2007, Monsoon Multimedia was sued by Software Freedom Conservancy on violating the open-sourcing term in GPL [2, 3]. It was regarded as the first lawsuit of GPL violation since it was published 18 years ago. In 2008, in the legal litigation of Jacobsen v. Katzer [7], the U.S. Federal Circuit Court of Appeals established that license terms from Artistic License 1.0 are enforceable copyright conditions under both federal copyright law and state contract law. It was a landmark victory for open source movement and had a great impact on open source movement [12]. Nowadays, the consequences of violating open source license are well-known, thanks to dozens of open source license litigation cases [8]. License terms are enforceable copyright conditions and will cause consequences if violated.

To tackle license violations in open source software, great effort has been devoted. On the one hand, for license incompatibility and inconsistency, the existing literature [25, 41, 50, 52, 67, 68, 70] proposes methods to identify license violations by comparing the differences in licenses between parties, e.g., the license of a system and the license of its dependencies or the license of a package and the licenses of its files. These methods do not necessarily require an in-depth understanding of license terms; rather, it focuses on identifying differences to determine incompatibilities or inconsistencies.

On the other hand, the license violation of non-compliance with the license's prescribed terms is understudied. The existing literature [19, 21, 31] concentrates on the well-known copyleft terms of open source licenses similar to the GPL [4]. However, the copyleft terms are only one of the many open source license terms. For example, the dispute of the legal case on Oracle America, Inc. and Rimini Street, Inc. in 2018 [15] was concentrated on distributing terms.

Currently, little attention has been paid to violations of modification terms (MTs). Considering the history of the open source movement, the open source license litigation will become more common in the future, including violations on MTs. Therefore, it is important to gain a better understanding of the MTs across open source licenses so that technical solutions can be proposed.

3 Understanding Modification Terms

We design an empirical study to understand modification terms (MTs) in open source licenses by answering four research questions.

- **RQ1 Modification Scope Analysis:** How do MTs define the scope of modification to the original work?
- **RQ2 Notice Content Analysis:** What are the required contents to put into the notice for obligating MTs?
- **RQ3 Notice Location Analysis:** How do MTs define the location where the required notice should be put?
- **RQ4 Obligation Analysis:** What is the obligation of MTs in open source licenses?

To answer these research questions, we collect open source licenses that declare MTs, and label each MT with respect to modification scope, notice content, and notice location. Then, we model the obligations of MTs with the above three dimensions.

3.1 Collecting and Labeling MTs

To understand MTs in open source licenses, we first collect a total of 107 approved licenses which are listed on Open Source Initiative (OSI) [36]. OSI is a non-profit corporation actively involving in and advocating open source. These licenses include open source licenses such as Apache-2.0, BSD and MIT. Then, we hire two graduate students who major in Law as experts to read each license document to find MTs and obligations in the MTs. They resolve conflicts and reach a Cohen's Kappa coefficient of 0.887.

9 of the 107 open source licenses do not mention modifications. 40 open source licenses mention modifications, but do not require any obligation on modifications. For example, the MIT license does not require further obligation on modifications. It dictates that *"Permission is hereby granted, free of charge, ... to deal in the Software without restriction, including without limitation the rights to use, copy,* ***modify****, merge, publish, ...".* 48 open source licenses declare MTs, and require certain forms of "notice" to describe modifications made to the original work as the obligation. 10 open source licenses mention modifications, and require other forms of obligation (e.g., putting trademarks).

To characterize MTs in these 48 licenses, the two hired experts discuss the obligations of MTs and extensively refer to online third-party resources when reading each MT following an open coding procedure [54]. Specifically, for each MT, they are required to find "what action will lead to MT violation?", "what action should be taken to avoid MT violation?", and "what are those actions in a concrete form?". After going through the 48 licenses, they reached an agreement on modeling the obligation of an MT from three dimensions, i.e., modification scope, notice content, and notice location. To label each MT with respect to these three dimensions, the two hired experts follow an open coding procedure [54] to inductively create composing elements and categories for the three dimensions by analyzing each MT. We use Cohen's Kappa coefficient to measure agreement, and it reached 0.760, 0.731 and 0.813, respectively. We respectively identified six, eleven, and twelve disagreements in labeling MT-related licenses, modification scope, and notice location. For example, eight of the disagreements on modification scope are related to understanding "software's source code files", i.e., whether to refer to all files or only source code-related files. Five of the disagreements on notice location originate from the misinterpretation of "cause any derivative works that you create to carry a prominent attribution notice"; i.e., whether to add a notice file for each modified file or each project. They resolve disagreements by referring to other licenses that provide more definition details or resorting to laws to cover a small but deterministic scope.

Table 1: Composing Elements of Modification Scope, Notice Content, and Notice Location

(a) RQ1: Modification Scope

ID	Composing Element
S_{c_1}	Source Code
S_{c_2}	Documentation
S_{c_3}	Configuration Files
S_{c_4}	Interface Definition Files
S_{c_5}	Scripts
S_{c_6}	Source Code Differential Comparison
S_{c_7}	Design Materials
S_{c_8}	Others

(b) RQ2: Notice Content

ID	Composing Element
C_{c_1}	Date
C_{c_2}	Author
C_{c_3}	Brief Statement
C_{c_4}	Informative Statement

(c) RQ3: Notice Location

ID	Composing Element
L_{c_1}	A Document
L_{c_2}	Each Modified File

3.2 Modeling Obligations of MTs

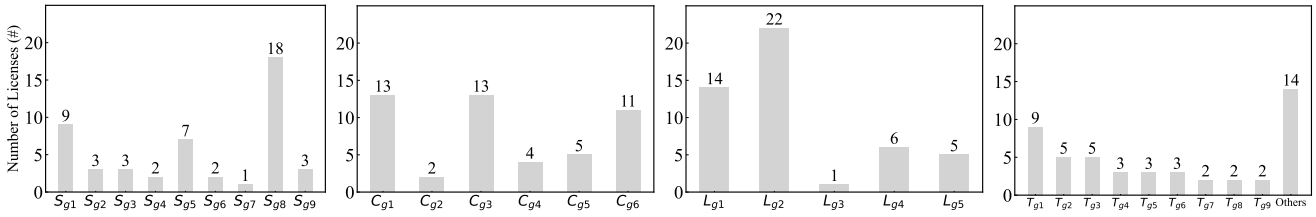
We model the obligations of MTs with modification scope, notice content, and notice location. Specifically, modification scope describes the types of original files that are modified (e.g., source code and documentation). Notice content represents the description of modifications required by the obligation of an MT. It is in the form of natural language statements with some meta-data (e.g., author). Notice location represents the location to put the notice content required by the obligation (e.g., a document). If users of an open source software modify the original files that are in the modification scope of an MT, they must put the required notice content to a required notice location to fulfill the obligation. Otherwise, the MT is violated. Open source licenses differ significantly in the obligation of MTs. We summarize and introduce composing elements and categories of each dimension (i.e., **RQ1**, **RQ2** and **RQ3**), and then present the obligation modeling based on these three dimensions (i.e., **RQ4**).

Modification Scope (RQ1). We summarize eight composing elements of modification scope (denoted as S_{c_i}, $1 \leq i \leq 8$), i.e., *Source Code*, *Documentation*, *Configuration Files*, *Interface Definition Files*, *Scripts*, *Source Code Differential Comparison*, *Design Materials*, and *Others*, as shown in Table 1a. The file types of S_{c_1} to S_{c_7} are easy to understand. We denote *Others* (S_{c_8}) to the rest of file types not listed in S_{c1} to S_{c7} because some licenses require obligation on modifications to all files.

Based on these composing elements, we categorize licenses into nine groups (denoted as S_{g_i}, $1 \leq i \leq 9$) with respect to the modification scope of their MTs, as presented in Table 2. The second column illustrates an example of MT partially excerpted from a license that belongs to each group. The third column reports the specific composing elements of the modification scope for each group. Notice

Table 2: RQ1: Groups of Modification Scope

Group	MT Excerpted from License Document	Composing Elements
S_{g_1}	**AFL-3.0**: The term "Source Code" means the preferred form of the Original Work for making modifications to it and all **available documentation describing how to modify the Original Work**.	S_{c_1}, S_{c_2}
S_{g_2}	**OHL-2.0**: 'Source' means information such as **design materials or digital code** which can be applied to Make or test a Product or to prepare a Product for use, Conveyance or sale, regardless of its medium or how it is expressed. It may include Notices.	S_{c_1}, S_{c_7}
S_{g_3}	**Apache-2.0**: "Source" form shall mean the preferred form for making modifications, including but not limited to **software source code**, **documentation source**, and **configuration files**.	$S_{c_1}, S_{c_2}, S_{c_3}$
S_{g_4}	**APSL-2.0**: "Source Code" means the human-readable form of a program or other work that is suitable for making modifications to it, including all modules it contains, plus **any associated interface definition files, scripts** used to control compilation and installation of an executable (object code).	$S_{c_1}, S_{c_4}, S_{c_5}$
S_{g_5}	**MPL-1.1**: "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any **associated interface definition files, scripts** used to control compilation and installation of an Executable, or **source code differential comparisons** against either the Original Code or another well known, available Covered Code of the Contributor's choice.	$S_{c_1}, S_{c_4}, S_{c_5}, S_{c_6}$
S_{g_6}	**RPL-1.5**: "Source Code" means the preferred form for making modifications to the Licensed Software and/or Your Extensions, including all modules contained therein, plus any **associated text, interface definition files, scripts** used to control compilation and installation of an executable program ...	$S_{c_1}, S_{c_2}, S_{c_4}, S_{c_5}$
S_{g_7}	**SPL-1.0**: "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any **associated documentation, interface definition files, scripts** used to control compilation and installation of an Executable, or **source code differential comparisons** against either the Original Code or another well known, available Covered Code of the Contributor's choice.	$S_{c_1}, S_{c_2}, S_{c_4}, S_{c_5}, S_{c_6}$
S_{g_8}	**GPL-3.0**: To "modify" a work means to copy from or adapt **all or part of the work** in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.	S_{c_1}–S_{c_8}
S_{g_9}	**LiLiQ-P-1.1**: "modified software": any modification made by a licensee to **one of the software's source code files, or any new source code file** that integrates the software or a substantial part of it	S_{c1}



(a) RQ1: Modification Scope **(b) RQ2: Notice Content** **(c) RQ3: Notice Location** **(d) RQ4: Obligation Group**

Figure 1: License Distribution across Groups of Modification Scope, Notice Content, Notice Location, and Obligation Group

that we use S_{c_1}–S_{c_8} to denote all files from the original work. *Source Code* (S_{c_1}) is included in the modification scope of all MTs.

Example 3.1. The MT of Apache-2.0 dictates that *"'Source' form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files"*. It belongs to S_{g_3} whose modification scope includes S_{c_1}, S_{c_2} and S_{c_3}. The MT of GPL-3.0 declares that *"To 'modify' a work means to copy from or adapt all or part of the work"*. Thus, we put it into S_{g_8} whose modification scope includes all files, i.e., S_{c_1} to S_{c_8}.

We analyze the distribution of 48 open source licenses across the nine groups of modification scope. The result is reported in Fig. 1a. 18 (37.5%) licenses belong to S_{g_8}, accounting for the largest portion of licenses. This indicates that nearly half of the licenses require obligation on modification to any file. 9 (18.8%) and 7 (14.6%) licenses respectively belong to S_{g_1} and S_{g_5}. Each of the rest six groups contains no more than three licenses.

Notice Content (RQ2). We summarize four composing elements of notice content (which are denoted as C_{c_i}, $1 \leq i \leq 4$), i.e., *Date*, *Author*, *Brief Statement* and *Informative Statement*, as listed in Table 1b. *Date* and *Author* describe when and who modifies the original files. We distinguish modification description into *Brief Statement* and *Informative Statement*. The former only requires you to state that you changed the files, while the latter also requires you to state how you changed the files, the detail of the change, etc.

Based on these composing elements, we categorize licenses into six groups (denoted as C_{g_i}, $1 \leq i \leq 6$) with respect to the notice content of their MTs, as shown in Table 3. Similar to Table 2, the second column shows an example of MT, and the third column lists the specific composing elements.

Example 3.2. The MT of LGPL-2.1 declares that "*You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change*". It belongs to C_{g1} as it requires to include *Date* and *Brief Statement* into notice content. The MT of SPL-1.0 states that "*... contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer ...*". It belongs to C_{g_6} as it requires to include *Date*, *Author* and *Informative Statement*.

We analyze the distribution of 48 open source licenses across the six groups of notice content. The result is shown in Fig. 1b. 13 (27.7%) licenses belong to C_{g_1} and C_{g_3}, respectively, and 11 (23.4%) licenses belong to C_{g_6}, which account for the largest portions of licenses. C_{g_3} only requires to add a brief statement, which is the simplest practice to obligate MTs among the six groups. Differently, C_{g_6} is the strictest practice to obligate MTs as it requires a date, author, and informative statement. Each of the rest three groups contains no more than five licenses.

Notice Location (RQ3). We summarize two composing elements of notice location (denoted as L_{c_i}, $1 \leq i \leq 2$), i.e., *A Document* and *Each Modified File*, as listed in Table 1c. L_{c_1} denotes that notice content is added to a separate file. L_{c_2} denotes that notice content is separately added to each modified file.

Based on the composing elements, we categorize licenses into five groups (denoted as L_{g_i}, $1 \leq i \leq 5$) with respect to the notice location of the MTs, as shown in Table 4. We use $L_{c_1} \mid L_{c_2}$ to denote that notice content can be added to either a document or each modified file, and `Unspecified` to denote the location is not clearly stated.

Table 3: RQ2: Groups of Notice Content

Group	MT Excerpted from License Document	Composing Elements
C_{g_1}	**LGPL-2.1**: You must cause the files modified to carry prominent notices **stating that you changed the files** and **the date of any change**.	C_{c_1}, C_{c_3}
C_{g_2}	**CDDL-1.0**: You must include a notice in each of Your Modifications that **identifies You as the Contributor of the Modification**.	C_{c_2}, C_{c_3}
C_{g_3}	**Apache-2.0**: You must cause any modified files to carry prominent notices **stating that You changed the files**.	C_{c_3}
C_{g_4}	**LPPL-1.3c**: Every component of the Derived Work contains prominent notices **detailing the nature of the changes to that component**, or a prominent reference to another file that is distributed as part of the Derived Work and that contains **a complete and accurate log of the changes**.	C_{c_4}
C_{g_5}	**OGTSL**: You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file **stating how** and **when you changed that file**...	C_{c_1}, C_{c_4}
C_{g_6}	**NASA-1.3**: Document any Modifications You make to the Licensed Software including **the nature of the change, the authors of the change**, and **the date of the change**.	$C_{c_1}, C_{c_2}, C_{c_4}$

Table 4: RQ3: Groups of Notice Location

Group	MT Excerpted from License Document	Composing Elements
L_{g_1}	**AFL-3.0**: You must cause all Covered Code to which You contribute to **contain a file documenting the changes** You made to create that Covered Code and the date of any change.	L_{c_1}
L_{g_2}	**Apache-2.0**: You must **cause any modified files to carry prominent notices** stating that You changed the files.	L_{c_2}
L_{g_3}	**RPL-1.1**: Document any Modifications You make to the Licensed Software including the nature of the change, the authors of the change, and the date of the change. This documentation must appear both **in the Source Code and in a text** file titled "CHANGES" distributed with the Licensed Software and Your Extensions.	L_{c_1}, L_{c_2}
L_{g_4}	**LPPL-1.3c**: Every component of the Derived Work contains **prominent notices** detailing the nature of the changes to that component, **or a prominent reference to another file** that is distributed as part of the Derived Work and that contains a complete and accurate log of the changes.	$L_{c_1} \mid L_{c_2}$
L_{g_5}	**GPL-3.0**: The work must carry **prominent notices** stating that you modified it, and giving a relevant date.	`Unspecified`

Example 3.3. The MT of AFL-3.0 states that "*You must ... contain a file documenting the changes You made to create that Covered Code and the date of any change*". It belongs to L_{g1} as it requires a file to document the changes. The MT of RPL-1.1 states that "*Document any Modifications You make ... This documentation must appear both in the Source Code and in a text file titled 'CHANGES' ...*". It belongs to L_{g3} as it requires adding notice content to both a file and each modified file. The MT of LPPL-1.3c states that "*Every component of the Derived Work contains prominent notices ... or a prominent reference to another file ... and that contains a complete and accurate log of the changes*". It belongs to L_{g_4} as it requires to add notice content to either a file or each modified file. The MT of GPL-3.0 states that "*The work must carry prominent notices stating that you modified it, and giving a relevant date*". It belongs to L_{g_5} as it does not clearly state a specific notice location.

We analyze the distribution of 48 open source licenses across the five groups of notice location. The result is shown in Fig. 1c. L_{g_2} and L_{g_1} occupy the largest portion of licenses, accounting for 45.8% (22) and 29.2% (14) of licenses, respectively. This indicates that it is the most common requirement to put notice either in a separate file or in each modified file. 5 (10.4%) licenses (belonging to L_{g_5}) have no specific requirement for notice location. Only 1 (2.1%) license (belonging to L_{g_3}) requires to put notice in both a separate file and each modified file. 6 (12.5%) licenses (belonging to L_{g_4}) require to put notice either in a separate file or each modified file.

Obligation Modeling (RQ4). For each of the 48 open source licenses, we model its MT t as a 2-tuple $\langle lic, O \rangle$, where lic denotes the license name, and O denotes the obligation of t. We model O as a 3-tuple $\langle S_{g_i}, C_{g_j}, L_{g_k} \rangle$, where S_{g_i}, C_{g_j} and L_{g_k} are the modification scope, notice content and notice location.

Based on the obligation, we categorize the 48 licenses into 23 groups (denoted as $T_{g_n}, 1 \leq n \leq 23$), where each group has the same obligation (i.e., S_{g_i}, C_{g_j} and L_{g_k}). Fig. 1d shows the distribution of 48 open source licenses across these groups. The nine groups containing the most licenses are separately reported, while the 14 groups containing only one license are put into *Others*. T_{g_1} requires S_{g_8}, C_{g_1} and L_{g_2}, accounting for 9 (18.8%) licenses. T_{g_2} requires S_{g_1}, C_{g_3} and L_{g_1}, containing 5 (10.4%) licenses. Overall, nearly 70.8% of the licenses declare the same obligation with at least one license.

4 Approach

Inspired by our empirical study in Sec. 3, we propose LiVo to automatically detect and fix violations of MTs during forking.

4.1 Approach Overview

Fig. 2 presents the approach overview of LiVo. It takes as inputs a base and a fork repository as well as the model of MTs for the 48 open source licenses (Sec. 3). LiVo works in three steps to detect and fix violations of MTs (i.e., the required notice is missing for the modified files that are in the modification scope) during forking. First, it detects obligating modifications of the fork repository (Sec. 4.2). Here we use commits as the granularity to recognize modifications. In that sense, we detect obligating commits that modify the original files that are in the modification scope of the MT of the base repository. Second, it extracts change logs from potential notice location in the fork repository (Sec. 4.3). These change logs contain potential notice for the obligating commits. Finally, it matches change logs with commit logs based on notice content of the MT to detect and fix MT violations for the fork repository (Sec. 4.4).

4.2 Obligating Modification Detection

This step recognizes the fork repository's modifications on the base repository that are under the obligation of the MT. Here, we choose commits as the granularity to detect such obligating modifications because commit logs provide sufficient semantic information about modifications which eases the detection and fixing of MT violations.

4.2.1 Obligating File Detection. After forking occurs, the base repository B_1 is evolved into the latest base repository B_2. Meanwhile, the fork repository evolved from B_1 into F. The modifications between B_1 and F may change files in the base repository which are protected by MTs. This stage builds file mappings between the base and fork repository to denote file reuse relations, detects the license of the file in the base repository, and finally selects obligating files protected by the license.

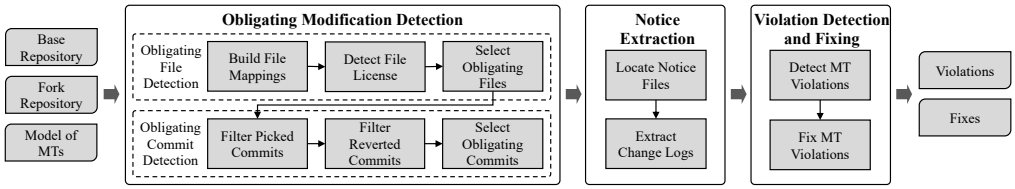


Figure 2: Approach Overview of LiVo

Build File Mappings. First, we build file mappings between the base and fork repository. Each file mapping is denoted as $\langle f, f' \rangle$, where $f \in B_1 \cup B_2$, $f' \in F$, and f' reuses f. Specifically, we build file mappings based on three reuse scenarios. First, for each file $f' \in F$, if there exists $f \in B_1 \cup B_2$ that has the same path name and file name with f' and is created by the base repository, we build a mapping $\langle f, f' \rangle$. It indicates that the fork repository reuses f and does not move or rename f. Here we also consider files in B_2 because the fork repository may pick or merge commits from the base repository. Second, for each of the remaining files f' in F, we backtrack the commit history, use the rename detector provided by JGit [20] to locate the file f in the base repository that is renamed into f', and build a mapping $\langle f, f' \rangle$. It indicates that the fork repository reuses f and moves or renames f. Third, for the rest files in F, we use Saga [45] to detect file-level clones against B_1 and B_2. If $f' \in F$ has a clone relation with $f \in B_1 \cup B_2$ and f is created by the base repository, we build a mapping $\langle f, f' \rangle$. It indicates that the fork repository reuses the code in f by copy-and-paste.

Detect File License. Then, for each file mapping $\langle f, f' \rangle$, we apply Ninka [27] to detect the file license of f, which extracts text from file header and identifies the license by matching extracted text against predefined license templates. If Ninka reports an unknown result (i.e., no license is matched), we apply Ninka to identify project license, and use it as the file license. We use lic_f to denote the detected file license of f.

Select Obligating Files. Last, for each file mapping $\langle f, f' \rangle$, if the detected file license lic_f belongs to the 48 open source licenses, we determine whether f is protected by the MT t of lic_f ($t.lic = lic_f$), i.e., whether f is in the modification scope of $t.lic$. To this end, we distinguish S_{c_1} to S_{c_7} with file extensions. Specifically, we first search FileInfo.com (which is a database of over 10,000 file extensions with detailed information about the associated file types) with names of S_{c_1} to S_{c_7} as queries. Then, we manually check and correct the query results. Finally, we map 2,085 file extensions to S_{c_1} to S_{c_7}, and provide the detailed mappings at our website [56]. Given such mappings, if the file extension of f belongs to the corresponding file extensions of the modification scope (i.e., $t.O.S_{g_i}$), we select f as the obligating file that is protected by t. We denote all obligating files as F_{ob}, and denote each obligating file as a tuple $\langle f, f', t \rangle$, meaning that the obligating file f is under the protection of t and the modifications in f' made on f should fulfill t.

4.2.2 Obligating Commit Detection. We denote the commit history of the base repository between B_1 and B_2 as H_B, denote the commit history of the fork repository between B_1 and F as H_F, and denote each commit as a tuple $\langle id, a, d, msg \rangle$, where id denotes the commit SHA id, a denotes the author, d denotes the date of the commit, and msg denotes the commit log. This stage first filters picked and reverted commits (which are free from the obligation) from H_F, and then selects obligating commits (which modify the obligating files in F_{ob}) from H_F.

Filter Picked Commits. The fork repository might pick or merge commits from the base repository. Such commits are submitted by the base repository, and hence are free from the obligation. Therefore, we need to filter such commits from H_F. In particular, for each commit $h \in H_F$, if there exists $h' \in H_B$ that has the same SHA id with h (i.e., $h.id = h'.id$), we remove h from H_F (i.e., $H_F = H_F - \{h\}$).

Filter Reverted Commits. The `git revert` command is used to revert changes introduced by a commit h and append a new commit h' with the resulting reverse content. As h and h' do not produce any changes in the latest fork repository, we need to filter these reverted commits from H_F. Specifically, the commit log of h' is automatically generated by the revert command with a pattern of "Revert $h.msg$". We use this pattern to identify the reverted commits h and h', and remove them from H_F (i.e., $H_F = H_F - \{h, h'\}$).

Select Obligating Commits. After filtering picked and reverted commits, we start with the latest commit $h \in H_F$ and determine whether h is an obligating commit by the following procedure. We get the changed files in h, and iterate each changed file. If a changed file f_c is added, deleted or modified and there exists an obligating file $f_{ob} \in F_{ob}$ such that $f_c = f_{ob}.f'$ (i.e., h modifies the obligating file), we regard h as an obligating commit, and add the required MT $f_{ob}.t$ to T_h. If a changed file is renamed from f_c to f'_c and there exists an obligating file $f_{ob} \in F_{ob}$ such that $f'_c = f_{ob}.f'$ (i.e., h renames the obligating file), we regard h as an obligating commit, add the required MT $f_{ob}.t$ to T_h, and add $\langle f_{ob}.f, f_c, f_{ob}.t \rangle$ to F_{ob} to continue the tracking of earlier changes on f_c. Then, we continue the above procedure on h's previous commit in H_F. Finally, we denote the set of identified obligating commits as H_{ob}, while the other commits between B_1 and F are considered obligation-free (`OF`).

For each obligating commit h, T_h stores all the required MTs because h might modify multiple obligating files protected by different MTs. Therefore, we conduct a union operation on the notice content and notice location required by the MTs in T_h, as formulated by Eq. 1, where C_h and L_h respectively denote the required notice content and notice location for h.

$$C_h = \cup_t^{T_h}(t.O.C_{g_j}),\ L_h = \cup_t^{T_h}(t.O.L_{g_k}) \tag{1}$$

For example, given an obligating commit that changes two obligating files f_1 and f_2, f_1 conforms to t_1, and f_2 conforms to t_2. t_1 requires C_{c_3} as the notice content, and t_2 requires C_{c_1} and C_{c_4} as the notice content. Thus, the required notice content for this obligating commit is C_{c_1}, C_{c_3} and C_{c_4}.

4.3 Notice Extraction

This step locates notice files in F, and extracts change logs which could be noticed for obligating commits.

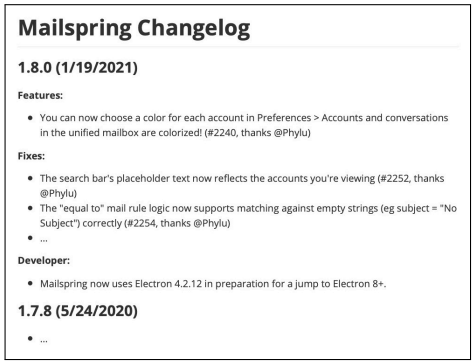


Figure 3: An Illustrative Example of Notice Files

Locate Notice Files. We locate notice files in F based on our understanding of notice location (see Sec. 3.2) which can be either a separate document (i.e., L_{c_1}) or each modified file (i.e., L_{c_2}). On the one hand, regarding L_{c_1}, we search files in F with certain file name patterns, e.g., release.txt, ChangeLog.txt. We provide a full list of file name patterns at our website [56].

On the other hand, we do not analyze files regarding L_{c_2} due to our pilot analysis of the dataset. In repository collection, we examined a total of 53,982 obligating commits from 178 fork repositories (see Sec. 5.1). Among these, there were merely 306 obligating commits that modified at least one source file through both textual changes and code changes, accounting for 0.58% of the total commits. The textual changes can be summarized into six types; i.e., software version and file coding change (99, 32.4%), license replacement (27, 8.8%), copyright author change (81, 26.5%), copyright year change (61, 19.9%), mismarked change due to git diff (29, 9.5%), and update date change (9, 2.9%). All these types cannot be regarded as an obligated notice. Therefore, we conclude that developers typically do not put notice in each modified file. Thus, we relax the obligation of adding a notice to each modified file to allow for notice in a separate document. Furthermore, we find that there is zero matching change log in a notice file from 306 obligating commits which should conform to L_{g_2}. Therefore, the obligating commits that should comply with L_{g_2} would still be detected using this relaxation because those commits also do not comply with L_{g_1}.

Extract Change Logs. After we locate notice files, we parse and extract change logs. Generally, change logs are written in semi-structured texts whose extraction can be automated based on templates. To derive the templates, we manually inspect 519 fork repositories from the 1,747 fork repositories with a confidence level of 99% and a margin of error of 4.76%. The 1,747 fork repositories have over 300 stars and have development activities (e.g., commits) after October, 2018. We successfully locate notice files in 100 fork repositories. Specifically, most of them (i.e., 91 fork repositories) have change logs organized in blocks. Each block consists of a subtitle and a list of change logs. The subtitle contains date literals in 55 fork repositories and has only version literals in the rest 36 fork repositories. Fig. 3 shows a clip of notice file in Mailspring-Libre[51], where the subtitle contains both date and version literals. Besides, the rest 9 have short change logs which are not separated by subtitles.

For notice files where change logs are separated by subtitles, we iterate each line for date expressions, which are matched by the pattern in Fig. 4. Each date expression is the starting point of a block of

```
Date  := {Year}{SEP}{Month}{SEP}{Day} | {Month}{SEP}{Day}{SEP}{Year} |
         {Day}{SEP}{Month}{SEP}{Year}
SEP   := (SPACE, -, ., /)
Year  := \d{4}
Month := \d{1,2} | (jan., ..., dec.) | (january, ..., december)
Day   := \d{1,2}((st|nd|rd|th)?)
```

Figure 4: The Pattern of Date

change logs. Each block is associated with a range of date, i.e., from the starting date d_s to the ending date d_e, indicating that the modifications described by the change logs in the block occur between d_s and d_e. Here, d_e is extracted from the current block, while d_s is extracted from the next block. For example, d_s and d_e of the first block in Fig. 3 are respectively 2020-05-24 and 2021-01-19. Then, we consider each non-blank line *msg* in the block as a change log. For each non-blank line, we search for author $a \in A$ (A is obtained by collecting author from each commit in H^F before filtering). For example, Phylu is the author for three change logs in Fig. 3. For other notice files (i.e., with subtitles that do not contain data expressions, and with no subtitles), we consider each non-blank line as a change log in the same way as the above procedure.

We denote a change log n as a tuple $\langle msg, d_s, d_e, a \rangle$, and the whole set of change logs as N. d_s, d_e and a might be blank when there is no date and author declaration in change logs. Note that the extracted change logs might be noisy. For example, the non-blank line "Features" in Fig. 3 is also considered as a change log. Besides, some change logs describe the changes in fork repositories' files. However, such noises will not affect the accuracy of LiVo because they will not be matched to any obligating commits.

4.4 Violation Detection and Fixing

Detect MT Violations. For each obligating commit $h \in H_{ob}$, we try to find a change log $n \in N$ such that n fulfills the obligation required by the modification in h. If such a change log is not found, we detect an MT violation for h. Specifically, if the notice content C_{c_3} or C_{c_4} is required, i.e., $C_{c_3} \in C_h$ or $C_{c_4} \in C_h$ (in fact every MT of the 48 open source licenses requires C_{c_3} or C_{c_4}), we measure the similarity between the commit log $h.msg$ and the change log $n.msg$. If the similarity exceeds a threshold th, the obligating commit h is regarded to fulfill C_{c_3} or C_{c_4} through the change log n. If such a change log is not found, we detect an MT violation for h due to missing notice (VN). Here, we do not distinguish between C_{c_3} (*brief statement*) and C_{c_4} (*informative statement*) as the boundary between brief and informative natural language texts lacks a clear definition. We use TF-IDF [38, 47] to measure the similarity between $h.msg$ and $n.msg$. $h.msg$ and $n.msg$ are represented by a term frequency vector, and the document collection is the list of change logs in N. We implement the TF-IDF by using *TfidfTransformer* from *scikit-learn* [53].

Then, if such a change log n is found and the notice content C_{c_1} is required (i.e., $C_{c_1} \in C_h$), we compare the date of h with the date range of n. If $h.d$ is within the date range of $n.d_s$ and $n.d_e$, the obligating commit h is considered to fulfill C_{c_1}. Otherwise, we detect an MT violation due to missing date (VD).

Finally, if such a change log n is found and the notice content C_{c_2} is required (i.e., $C_{c_2} \in C_h$), we compare the author of h with the author of n. If $h.a$ is the same as $n.a$, the obligating commit h is considered to fulfill C_{c_2}. Otherwise, we detect an MT violation due to missing author (VA). If h fulfills all the obligations in C_h, it is considered fully obligated (OB). Our detection result for each

obligating commit $h \in H_{ob}$ is denoted as a tuple $\langle h, n, type \rangle$, where h denotes the obligating commit, n denotes the matched change log, and $type$ denotes the fulfillment type which can either be violated due to missing notice (VN), violated due to missing date (VD), violated due to missing author (VA), or fully obligated (OB).

Fix MT Violations. For an obligating commit h whose fulfillment type is VN, we create a new change log n such that $n.msg = h.msg$, $n.d_s = h.d$, $n.d_e = h.d$ and $n.a = h.a$. For an obligating commit h whose fulfillment type is VD (resp. VA), we update the matched change log n such that $n.d_s = h.d$ and $n.d_e = h.d$ (resp. $n.a = h.a$). Finally, we write the newly added or updated change log to the notice file. If the notice file does not exist, we create a new notice file.

5 Evaluation

We have implemented LiVo with 4.2K lines of Java code and 0.4K lines of Python code[56].

5.1 Evaluation Setup

We design three research questions.

- **RQ5 MT Prevalence Evaluation**: How is the prevalence of MTs in real-world fork repositories? (see Sec. 5.2)
- **RQ6 MT Violation Evaluation**: How is the violation of MTs in real-world fork repositories? (see Sec. 5.3)
- **RQ7 Effectiveness and Efficiency Evaluation**: How is the effectiveness and efficiency of LiVo? (see Sec. 5.4)

Repository Collection. We use Github GraphQL Explorer to query and collect pairs of base and fork repositories. Specifically, we query for fork repositories that have over 300 stars and have development activities (e.g., commits) after October 2018. Our query returns 1,743 fork repositories with their corresponding base repositories. Then, we identify 814 fork repositories whose licenses are within the list of the 107 licenses in our study in Sec. 3. Of the rest 929 repositories, 884 repositories do not declare any license, and 45 repositories declare *unlicense* [65] or other licenses which are out of the scope of our 107 licenses. Further, we filter 117 fork repositories from the 814 fork repositories because 84 of them indicate through ReadMes or merged commits that they have collaboration relations with the base repositories, 12 of them are no longer maintained, and 21 of them do not disclose the organization information so that the relation of developers between the base and fork developers is unknown. This results in 697 fork repositories. Finally, we obtain 178 fork repositories whose licenses are within the scope of our 48 open source licenses while the rest are not.

We further manually locate and analyze notice files in these 178 fork repositories. We find that 57 (32.0%) of these 178 fork repositories contain notice files, which is relatively small. These 57 fork repositories have an overlap of 18 with the sampled 519 fork repositories in Sec. 4.3. All of these 57 fork repositories have change logs organized in blocks. Among them, 48 fork repositories have subtitles inclusive of date expressions, and 10 fork repositories have subtitles absent of date expressions.

We also manually extract change logs in these 57 fork repositories, rank the fork repositories in the descending order of the number of change logs, and plot them in Fig. 5. 70% of the 57 fork repositories have less than 551 change logs. The maximum number of change logs in a single fork repository is 3,281, but this

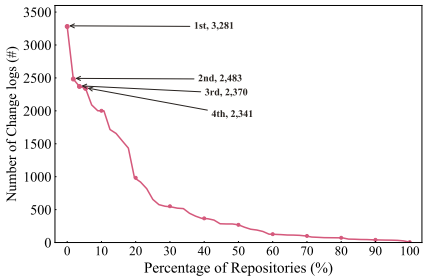


Figure 5: Number of Change Logs in Fork Repositories

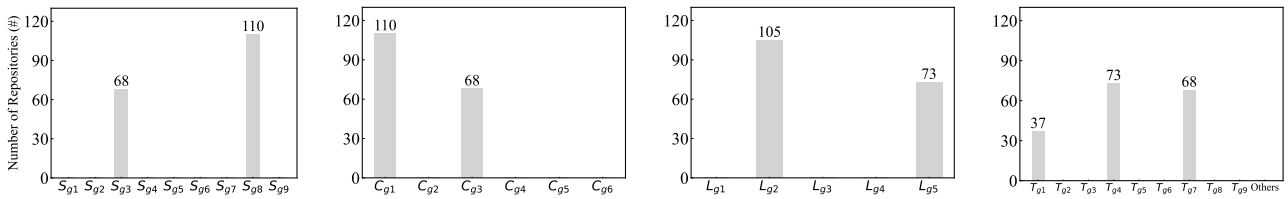
number decreases drastically in the following repositories, with 2,483 change logs in the second, 2,370 change logs in the third, and 2,341 change logs in the fourth fork repository. These four fork repositories contribute 30.4% of the 34,487 change logs.

RQ Setup. To answer **RQ5**, we analyze the distribution of modification scope, notice content, notice location, and obligation group across the 178 base repositories according to the licenses detected by Ninka. To answer **RQ6**, we run LiVo against each pair of base and fork repositories, and obtain the detection result of MT violations. Specifically, we measure the number of repositories and commits that violate MTs. To answer **RQ7**, we evaluate the accuracy, usefulness, and performance of LiVo. For accuracy, we manually construct the ground truth and measure the detection accuracy of LiVo on different fulfillment types. Since it is a multi-class classification problem, we use macro-precision and macro-recall [30] as the accuracy metrics. Further, we measure the accuracy sensitivity of LiVo to the configurable similarity threshold th. For usefulness, we submit pull requests with detailed reference on violations and fixes to the corresponding fork repositories. For performance, we measure the time overhead of LiVo on each repository pair.

Ground Truth Construction. We randomly sample and manually check commits from a total of 176,273 commits in the 178 fork repositories until we successfully identify 100 commits for each fulfillment type (i.e., OB, VN and VD) as well as 100 commits that are obligation-free (i.e., OF). We inspect both commit messages and code changes in the sampled commits. By understanding the meaning of commit messages and corresponding code elements, we established mappings to change logs. We fail to identify any commit for VA because no repository requires C_{c_2} (i.e., author) in their licenses. We also fail to observe one-commit-to-many-change-logs mappings. However, we observe 100 many-commits-to-one-change-log mapping commits in our ground truth, which can reflect how capable LiVo can handle such scenario. Finally, we sample a total of 39,022 commits, and construct a ground truth for 400 commits, achieving a confidence level of 95% and a margin error of 4.9%. Specifically, two of the authors independently determine the fulfillment type for each sampled commit and a third author was involved to resolve disagreements. We use Cohen's Kappa coefficient to measure agreement, and it reaches 0.891. A third author is involved in resolving disagreements. Notice that the ground truth is manually constructed in two person-months.

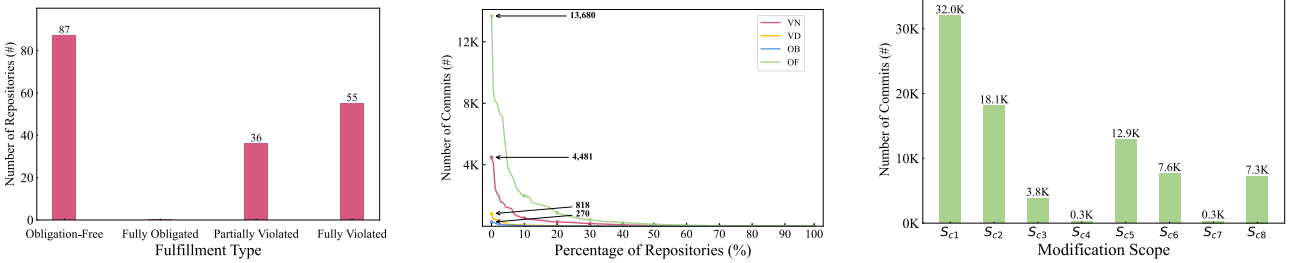
5.2 Real-World MT Prevalence (RQ5)

As revealed in Sec. 5.1, 178 of the 814 highly-starred forked repositories require MTs in their licenses, which stands for a non-negligible proportion. In that sense, it is useful to detect and fix MT violations



(a) Modification Scope (b) Notice Content (c) Notice Location (d) Obligation Group

Figure 6: Number of Repositories w.r.t Modification Scope, Notice Content, Notice Location, and Obligation Group



(a) Repositories That Violate MTs (b) Commits That Violate MTs (c) Obligating Commits w.r.t Modification Scope

Figure 7: The Number of Repositories and Commits That Violate MTs

during forking. Surprisingly, these 178 base repositories only adopt five dominating licenses, i.e., Apache-2.0, GPL-3.0, GPL-2.0, AGPL-3.0 and LGPL-2.1. These five licenses are respectively adopted in 68, 59, 32, 14 and 5 base repositories. Regarding MTs in these licenses, we report the number of repositories with respect to the three obligation dimensions (i.e., modification scope, notice content, and notice location) and the obligation groups in Fig. 6.

In particular, Apache-2.0 has the obligation $\langle S_{g_3}, C_{g_3}, L_{g_2} \rangle$, therefore contributing the bar with 68 repositories in Fig. 6a and Fig. 6b. The GPL family (i.e., GPL-3.0, GPL-2.0, AGPL-3.0 and LGPL-2.1) has the obligation of S_{g_8} and C_{g_1}, therefore contributing the bar with 110 repositories in Fig. 6a and Fig. 6b. Meanwhile, the GPL family is divided into two sub-groups, i.e., GPL-2.0 and LGPL-2.1 whose notice location is L_{g_2}, and GPL-3.0 and AGPL-3.0 whose notice location is L_{g_4}, respectively accounting for 37 and 73 repositories. As a result, the MTs fall into three obligation groups, as shown in Fig. 6d. T_{g_5} represents GPL-3.0 and AGPL-3.0, accounting for 73 repositories. T_{g_7} represents Apache-2.0, taking up 68 repositories. T_{g_1} represents GPL-2.0 and LGPL-2.1, accounting for 37 repositories.

Summary. 21.9% of the highly-starred forked repositories require MTs in their licenses, indicating the moderate prevalence of MTs during forking. Due to the concentrated adoption of five licenses, the MT obligation is concentrated in three groups.

5.3 Real-World MT Violations (RQ6)

Obligated/Violated Repositories. We categorize fork repositories into *Obligation-Free*, *Fully Obligated*, *Partially Violated* and *Fully Violated* repositories. The result is reported in Fig. 7a. *Obligation-Free* repositories mean that they have no obligating commit because they do not modify the original files protected by MTs, taking up 87 of the 178 fork repositories. *Fully Obligated* repositories have obligating commits and all of them fulfill the MTs by corresponding change logs. Unfortunately, we find zero fork repository that falls into this category. *Partially Violated* repositories have obligating commits and part of them violate the MTs, taking up 36 of the 178 fork repositories. *Fully Violated* repositories have all of their obligating commits violate the MTs, taking up 55 of the 178 fork repositories. Overall, 91 (51.1%) fork repositories violate the MTs.

Obligated/Violated Commits. LiVo detects 53,982 obligating commits from the 178 fork repositories, which is significantly larger than the total number of 34,487 change logs (as reported in Sec. 5.1). Finally, LiVo detects a total of 51,435 MT violations, i.e., 51,435 commits that violate the MTs. 4.7% obligating commits fulfill MTs.

We rank the fork repositories in the descending order of the number of commits that are obligation-free (OF), fully obligated (OB), violated due to missing notice (VN), and violated due to missing date (VD), shown in Fig. 7b. LiVo does not detect any commit that is violated due to missing author (VA) as no repository requires C_{c_2} (i.e., author) in their licenses. The number of obligation-free commits (OF) significantly exceeds the number of obligating commits (i.e., OB, VN and VD). This is reasonable because fork repositories extend base repositories by adding new features. Among the obligating commits, there are respectively 45,470 and 5,965 violated commits due to missing notice (VN) and missing date (VD), 18 and 2 times larger than the 2,547 obligated commits (OB). For example, *INTI-CMNB/KiBot* has the largest number of obligated commits (OB), which is 270. Contrarily, the number of violated commits due to missing notice (VN) and missing date (VD) reaches their peak at 4,481 in *SonixQMK/qmk_firmware* and 818 in *jobobby04/TachiyomiSY*.

Obligating Commits w.r.t Modification Scope. Fig. 7c presents the number of obligating commits with respect to modification scope. Specifically, source code (S_{c_1}) is modified in 32.0K (59.4%) commits, followed by documentation (S_{c_2}) and scripts.

Summary. MT violations are quite severe. 51.1% of the fork repositories violate MTs. 95.3% of the obligating commits violate MTs. 59.4% of the obligating commits contain modification to source code (S_{c_1}), which is the most prevalent element.

5.4 Effectiveness and Efficiency of LiVo (RQ7)

Accuracy. Using our ground truth, we report the accuracy of LiVo in detecting the four types of commits (i.e., VN, VD, OF and OB) in Table 5. We use the multi-class measure of precision and recall provided in *scikit-learn* [53] to compute the result. Overall, LiVo

Table 5: Accuracy Results

Metric	VN	VD	OF	OB	Macro-Average
Pre.	0.67	0.69	1.00	0.92	0.82
Rec.	0.76	0.78	0.93	0.72	0.80

achieves a macro-precision of 0.82 and macro-recall of 0.80. Moreover, we analyze the 81 mislabeled commits. There are 38 FPs (false positives) and 24 FNs (false negatives) for detecting VN commits, 35 FPs and 22 FNs for detecting VD commits, 0 FP and 7 FNs for detecting OF commits, and 8 FPs and 28 FNs for detecting OB commits. FPs and FNs can occur for one commit in multi-class classification.

Specifically, 7 OF commits are incorrectly labeled because the result of license detection is incorrect. We classify the other 74 mislabeled commits into three categories. First, 21 and 3 VN commits are incorrectly labeled into VD and OB, respectively. It causes most of the false negatives for detecting VN. The reason is due to the semantic unawareness of TF-IDF. For example, the identifier in the commit message usually represents a commit or an issue, and it also commonly serves as an identifier of other domains. The matching process ignores the semantic representation of the identifier, and assigns a higher weight to it because it is rarely seen. Second, 19 VD commits are incorrectly labeled into VN, and 15 OB commits are incorrectly labeled into VN. It causes most of the false positives for detecting VN. The reason is that those commit logs are common and meaningless, e.g., "fixing bugs" and "updating files". Although there are file, class, or method names in some of those commits, the names are very common, which causes their assigned weight in TF-IDF to be lower, e.g., "RELEASE.md", "run()". Third, 3 VD commits are incorrectly labeled into OB, and 13 OB commits are incorrectly labeled into VD, caused by the unsound extraction of date.

Further, we analyze the sensitivity of the accuracy of LiVo to the similarity threshold th. Specifically, we configure th from 0.1 to 0.9 by a step of 0.1, and report the result in Fig. 8. LiVo achieves the best macro-precision and macro-recall when th is set to 0.3. As th increases from 0.3, macro-precision is relatively stable, whereas macro-recall suffers a decreasing trend. Therefore, LiVo is sensitive to th, and we believe 0.3 is empirically a good value for th. Notice that we report the result in Table 5 when th is set to 0.3.

Usefulness. To evaluate the practical usefulness of LiVo, we submit 91 pull requests, using the violations and fixes generated by LiVo, to the violated repositories which have active development activities recently in order to obtain quick feedback. So far, we have received responses from 44 fork repositories.

In particular, 18 fork repositories give positive responses in the corresponding pull requests. 8 fork repositories approve our pull requests and merge them into the main branch. 2 fork repositories fix the violations themselves using a separate commit, while 1 fork repository is immediately archived to be read-only. 4 fork repositories claim that they are authors from the base repositories and hence there is no need to conform to such obligations. 3 fork repositories respond positively but do not take further actions so far.

Despite the positive responses, we also receive unwillingness from 26 fork repositories. 7 of them directly close the pull requests without further comments, while 4 of them discuss the pull requests but do not reveal their attitude or actions. 15 of them refuse to do further actions. Specifically, 5 of them think that the commit history itself reflects the fulfillment of MTs and thus there is no violation. Interestingly, although 2 of them refuse, they choose to delete or archive their repositories potentially to avoid legal risks. 4 of them disagree with our violation definition. Particularly, the first one simply replies with a "No" without any further comment. The second one considers the statement of forking the base repository in the ReadMe file as fulfilling the MTs. However, the statement fails to describe the required notice content. The third one claims that only the original authors of the base repository can request to obligate the MTs. The last one disagrees with the violated commits. Besides, 4 of them consider our pull requests as spam.

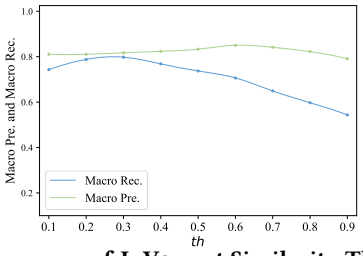


Figure 8: Accuracy of LiVo w.r.t Similarity Threshold th

Performance. To evaluate the performance of LiVo, we measure the time cost for each pair of repositories. LiVo takes 72.2 seconds in median and 229.8 seconds on average for each repository pair. For half of the repository pairs, LiVo consumes 32.4 seconds to 190.7 seconds. The minimal time cost is 8.9 seconds, while the maximal time cost is 4481.8 seconds. Notice that our obligating modification detection step accounts for approximately 92.3% of the overall time cost, primarily attributed to intensive Git operations.

Summary. LiVo achieves a macro-precision of 0.82 and a macro-recall of 0.80. 18 pull requests have received positive responses, and 8 of them have been merged. These results demonstrate the effectiveness of LiVo. Besides, LiVo takes averagely 229.8 seconds to analyze one repository pair, demonstrating its efficiency.

5.5 Discussion

Threats. The main threat to our empirical study is the labeling process. It involves manual interpretation from the two hired experts who major in Law. The license texts written in natural language might bring ambiguity. To mitigate it, they discuss disagreements or uncertainty and extensively refer to online resources to reach a consensus. Another threat to our evaluation is the ground truth construction. First, the scale is not large due to the expensive manual efforts. However, the scale ensures an acceptable confidence level. Second, the quality of the ground truth might affect our effectiveness evaluation. We mitigate it by following open coding procedures to build the ground truth. Besides, our approach did not consider change logs in modified files because they rarely existed.

Limitations. The limitations of our work are four-folds. First, our model of MT obligations is not exhaustive to all licenses. For example, there are 567 licenses from SPDX [9], and 127 of them are included in OSI. These new licenses can be incorporated into LiVo by understanding their modification scope, notice content and notice location, without any changes to our implementation. We plan to include more licenses into our model. Second, LiVo takes base and fork repositories with their commit history as inputs. The violation may be more prevalent and severe when commit history is

not publicly available. We plan to extend LiVo to use software composition analysis (SCA) to detect obligating files, and code change summarization to generate logs. Third, as revealed by our accuracy evaluation, the main inaccuracy is caused by semantic unawareness between *h.msg* and *n.msg* due to the limitations of TF-IDF. We plan to leverage semantic-aware techniques to further improve the accuracy of LiVo. Finally, we currently employ some relaxation to not distinguish between C_{c_3} and C_{c_4} and between L_{c_1} and L_{c_2}. We plan to resort to license regulators for formal interpretations.

6 Related Work

License Identification. The first step is to automatically identify licenses from text or code files. Tuunanen et al. [57] develop ASLA to identify licenses based on regular expressions. ASLA shows a competitive performance to two open source license analyzers, i.e., OSLC [16] and FOSSology [22, 28]. However, they do not report unknown licenses whose regular expressions are not prepared. To address this problem, German et al. [27] design Ninka to first break license statement into sentences and then match sentence-tokens with sentence-tokens already manually identified for each license (i.e., a license rule) to identify licenses. Ninka will report an unknown license if no license is matched. To help generate license rules for licenses reported as unknown by Ninka, Higashi et al. [32] leverage a hierarchical clustering to group unknown licenses into clusters of files with a single license. Besides, Vendome et al. [62] propose a machine learning-based approach to identify exceptions appended to licenses. Kapitsaki and Paschalides [42] design FOSS-LTE to identify license terms from license texts. Our work uses Ninka to identify declared licenses in each file.

License Understanding. Manabe et al. [48] analyze license changes along with the evolution of four open source systems. Similarly, Di Penta et al. [18] automatically tracks license term changes in six open source systems. These two studies reveal that licenses change frequently and substantially. Vendome et al. [59, 61] investigate when and why developers adopt and change licenses by analyzing commits and surveying developers. Almeida et al. [11] explore whether developers understand the licenses they use. To the best of our knowledge, our work is the first to systematically characterize the modification terms in open source licenses.

License Bug Detection. Vendome et al. [60] characterize license bugs by building a catalog and understanding their implications on the software projects they affect. Several approaches have also been proposed to detect license bugs (or license violations), which consist of license incompatibility and inconsistency.

License incompatibility refers to the incompatibility among the licenses of a system and its declared dependencies or reused source code. German et al. [26] propose a concept model to formally specify licenses and detect incompatibilities among licenses of a system and its declared dependencies. Mathur et al. [49] and Golubev et al. [29] empirically investigate incompatibilities among licenses of a system and its reused code via manual analysis. German and Di Penta [23] propose a semiautomatic approach Kenen to open source license compliance for Java systems.Similarly, van der Burg [58] propose to automatically identify the dependencies of a system by analyzing the build processKapitsaki et al. [41, 52] model license compatibility into a directed graph, which automates license incompatibility detection with Software Package Data Exchange. Xu et al. [70] leverage natural language processing to identify license terms and infer rights and obligations for incompatibility detection without any prior knowledge about licenses. Hemel et al. [31], Duan et al. [19] and Feng et al. [21] identify open source components through binary analysis, and detect violations of GPL licenses which require the binary to be open source. Xu et al. [71] studies license incompatibilities in the PyPI and proposed a SMT-solver-based approach to remediate incompatibilities.

License inconsistency specifically refers to the inconsistency between the licenses of two source files that are evolved from the same provenance, and the inconsistency between the license of a package and the license of each file of the package. German et al. [25] and Wu et al. [67, 68] focus on the first case. They first use clone detection to find source files that have code clones and use Ninka to identify licenses of these source files for inconsistency detection. German et al. [24], Di Penta et al. [17] and Mlouki et al. [50] focus on the second case. They either use metadata or tools like FOSSology and Ninka to identify licenses for inconsistency detection.

To prevent such license bugs, Liu et al. [46] design a learning-based approach to predict a compatible license.Kapitsaki and Charalambous [40] introduce a license recommender findOSSLicense to assist developers in choosing an appropriate open source license.

Different from these studies, we focus on a new type of license bugs, i.e., violations of modification terms in open source licenses. To the best of our knowledge, we are the first to detect and fix them for ensuring modification term compliance.

Open Source Software Reuse. Open source software reuse is a common activity in software development in various forms, e.g., copy and paste [39, 43, 44, 55], forking [13, 37, 73], and importing third-party dependencies [35, 63, 72]. It can cause various issues such as library conflicts [34, 64], API breaking [14, 33]. Our work is specifically focused on license bugs during forking. To the best of our knowledge, previous studies conduct empirical analysis of forking, but no work considers license bugs during forking. For example, Jiang et al. [37] explore developers' motivation and behavior on forking in GitHub. They find that some developers fork repositories to add new features. These activities should be conducted under the license modification terms. Brisson et al. [13] study the communication (e.g., pull requests, followers, and contributors) of repositories which are from the same software family via forking. Zhou et al. [73] analyze the forking history.

7 Conclusions

We model the MT with respect to modification scope, notice content and notice location in 107 open source license. We have designed LiVo to automatically detect and fix modification term violations during forking. Our evaluation results have demonstrated the severity of modification term violations and the effectiveness of LiVo.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant No. 62332005, 62372114, and 62402342), and "Sino-German Cooperation 2.0" Funding Program of Tongji University.

References

[1] 2022. *File With Apache 2.0 and My Modifications.* Retrieved March 21, 2023 from https://softwareengineering.stackexchange.com/questions/220068/file-with-apache-2-0-and-my-modifications

[2] 2022. *First lawsuit over GPL settled.* Retrieved March 21, 2023 from https://www.informationweek.com/software-services/first-lawsuit-over-gpl-settled-before-it-goes-to-court

[3] 2022. *First Open Source Gpl Violation Lawsuit Filed Us.* Retrieved March 21, 2023 from https://www.lvcriminaldefense.com/first-open-source-gpl-violation-lawsuit-filed-us/

[4] 2022. *GNU General Public License 3.0.* Retrieved March 21, 2023 from https://www.gnu.org/licenses/gpl-3.0.html

[5] 2022. *GNU Project.* Retrieved March 21, 2023 from https://en.wikipedia.org/wiki/GNU_Project

[6] 2022. *How to Specify Notices of All Changes in a Class.* Retrieved March 21, 2023 from https://opensource.stackexchange.com/questions/4419/aplv2-how-to-specify-notices-of-all-changes-in-a-class

[7] 2022. *Jacobsen v. Katzer.* Retrieved March 21, 2023 from https://en.wikipedia.org/wiki/Jacobsen_v._Katzer

[8] 2022. *Open Source License Litigation.* Retrieved March 21, 2023 from https://en.wikipedia.org/wiki/Open_source_license_litigation

[9] 2022. *SPDX License List.* Retrieved March 21, 2023 from https://spdx.org/licenses/

[10] 2022. *TVM Distribute Questions About Apache 2.0.* Retrieved March 21, 2023 from https://discuss.tvm.apache.org/t/tvm-distribute-questions-about-apache-2-0-license/8805/6

[11] D. A Almeida, G. C Murphy, G. Wilson, and M. Hoye. 2017. Do software developers understand open source licenses?. In *Proceedings of the IEEE/ACM 25th International Conference on Program Comprehension*. 1–11.

[12] R Michael Azzi. 2010. CPR: How Jacobsen V. Katzer resuscitated the open source movement. *U. Ill. L. Rev.* (2010), 1271.

[13] S. Brisson, E. Noei, and K. Lyons. 2020. We are family: analyzing communication in GitHub software repositories and their forks. In *Proceedings of the IEEE 27th International Conference on Software Analysis, Evolution and Reengineering*. 59–69.

[14] Aline Brito, Marco Tulio Valente, Laerte Xavier, and Andre Hora. 2020. You broke my code: understanding the motivations for breaking changes in APIs. *Empirical Software Engineering* 25 (2020), 1458–1492.

[15] casetext. 2023. *Oracle Inc. v. Rimini Street Inc.* Retrieved March 21, 2023 from https://casetext.com/case/oracle-usa-inc-v-rimini-st-inc#p962

[16] daxlec, devil_moon, lrontyne, sjkaaria, and villoks. 2007. *OSLC.* Retrieved March 21, 2023 from https://sourceforge.net/projects/oslc/

[17] M. Di Penta, D. M German, and G. Antoniol. 2010. Identifying licensing of jar archives using a code-search approach. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*. 151–160.

[18] 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*. 145–154.

[19] R. Duan, A. Bijlani, M. Xu, T. Kim, and W. Lee. 2017. Identifying open-source license violation and 1-day security risk at large scale. In *Proceedings of the ACM SIGSAC Conference on computer and communications security*. 2169–2185.

[20] Eclipse. 2023. *JGit.* Retrieved April 24, 2023 from https://www.eclipse.org/jgit/

[21] Muyue Feng, Weixuan Mao, Zimu Yuan, Yang Xiao, Gu Ban, Wei Wang, Shiyang Wang, Qian Tang, Jiahuan Xu, He Su, et al. 2019. Open-source license violations of binary software at large scale. In *Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*. 564–568.

[22] Linux Foundation. 2017. *Fossology.* Retrieved March 21, 2023 from https://www.fossology.org/

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

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

[25] Daniel M German, Massimiliano Di Penta, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2009. Code siblings: Technical and legal implications of copying code between applications. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories*. 81–90.

[26] Daniel M German and Ahmed E Hassan. 2009. License integration patterns: Addressing license mismatches in component-based development. In *Proceedings of the IEEE 31st international conference on software engineering*. 188–198.

[27] D. M German, Y. Manabe, and K. Inoue. 2010. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. 437–446.

[28] Robert Gobeille. 2008. The fossology project. In *Proceedings of the 2008 international working conference on Mining software repositories*. 47–50.

[29] Yaroslav Golubev, Maria Eliseeva, Nikita Povarov, and Timofey Bryksin. 2020. 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*. 54–64.

[30] Margherita Grandini, Enrico Bagli, and Giorgio Visani. 2020. Metrics for Multi-Class Classification: an Overview. arXiv:2008.05756 [stat.ML]

[31] A. Hemel, K. T. Kalleberg, R. Vermaas, and E. Dolstra. 2011. Finding software license violations through binary code clone detection. In *Proceedings of the 8th Working Conference on Mining Software Repositories*. 63–72.

[32] Y. Higashi, Y. Manabe, and M. Ohira. 2016. Clustering OSS license statements toward automatic generation of license rules. In *Proceedings of the 7th International Workshop on Empirical Software Engineering in Practice*. 30–35.

[33] Kaifeng Huang, Bihuan Chen, Linghao Pan, Shuai Wu, and Xin Peng. 2021. REPFINDER: Finding replacements for missing APIs in library update. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 266–278.

[34] Kaifeng Huang, Bihuan Chen, Bowen Shi, Ying Wang, Congying Xu, and Xin Peng. 2020. Interactive, effort-aware library version harmonization. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 518–529.

[35] K. Huang, B. Chen, C. Xu, Y. Wang, B. Shi, X. Peng, Y. Wu, and Y. Liu. 2022. Characterizing usages, updates and risks of third-party libraries in Java projects. *Empirical Software Engineering* 27, 4 (2022), 1–41.

[36] Open Source Initiative. 2022. *Open Source Licenses.* Retrieved March 21, 2023 from https://opensource.org/licenses

[37] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22, 1 (2017), 547–578.

[38] K. Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.

[39] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. 2009. Do code clones matter?. In *Proceedings of the IEEE 31st International Conference on Software Engineering*. 485–495.

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

[41] G. M Kapitsaki, F. Kramer, and N. D Tselikas. 2017. Automating the license compatibility process in open source software with SPDX. *Journal of systems and software* 131 (2017), 386–401.

[42] G. M Kapitsaki and D. Paschalides. 2017. Identifying terms in open source software license texts. In *Proceedings of the 24th Asia-Pacific Software Engineering Conference*. 540–545.

[43] M. Kim, L. Bergman, T. Lau, and D. Notkin. 2004. An ethnographic study of copy and paste programming practices in OOPL. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering*. 83–92.

[44] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. 2005. An empirical study of code clone genealogies. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*. 187–196.

[45] G. Li, Y. Wu, C. K Roy, J. Sun, X. Peng, N. Zhan, B. Hu, and J. Ma. 2020. SAGA: efficient and large-scale detection of near-miss clones with GPU acceleration. In *Proceedings of the IEEE 27th International Conference on Software Analysis, Evolution and Reengineering*. 272–283.

[46] X. Liu, L. Huang, J. Ge, and V. Ng. 2019. Predicting licenses for changed source code. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. 686–697.

[47] H. Luhn. 1957. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development* 1, 4 (1957), 309–317.

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

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

[50] O. Mlouki, F. Khomh, and G. Antoniol. 2016. On the detection of licenses violations in the android ecosystem. In *Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, Vol. 1. 382–392.

[51] notpushkin. 2023. *Changelog of Mailspring-Libre.* Retrieved March 21, 2023 from https://github.com/notpushkin/Mailspring-Libre/blob/master/CHANGELOG.md

[52] 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.

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[54] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.

[55] Wei Tang, Zhengzi Xu, Chengwei Liu, Jiahui Wu, Shouguo Yang, Yi Li, Ping Luo, and Yang Liu. 2022. Towards understanding third-party library dependency in

c/c++ ecosystem. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[56] LiVo. 2023. *The Replication Package of LiVo.* Retrieved March 21, 2023 from https://doi.org/10.5281/zenodo.7902495

[57] T. Tuunanen, J. Koskinen, and T. Kärkkäinen. 2009. Automated software license analysis. *Automated Software Engineering* 16, 3 (2009), 455–490.

[58] S. Van Der Burg, E. Dolstra, S. McIntosh, J. Davies, D. M German, and A. Hemel. 2014. Tracing software build processes to uncover license compliance inconsistencies. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 731–742.

[59] 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.

[60] C. Vendome, D. German, M. Di Penta, G. Bavota, M. Linares-Vásquez, and D. Poshyvanyk. 2018. To distribute or not to distribute? why licensing bugs matter. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering*. 268–279.

[61] C. Vendome, M Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk. 2015. License usage and changes: a large-scale study of java projects on github. In *Proceedings of the IEEE 23rd International Conference on Program Comprehension*. 218–228.

[62] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk. 2017. Machine learning-based detection of open source license exceptions. In *Proceedings of the IEEE/ACM 39th International Conference on Software Engineering*. 118–129.

[63] Ying Wang, Bihuan Chen, Kaifeng Huang, Bowen Shi, Congying Xu, Xin Peng, Yijian Wu, and Yang Liu. 2020. An empirical study of usages, updates and risks of third-party libraries in java projects. In *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution*. 35–45.

[64] Ying Wang, Ming Wen, Zhenwei Liu, Rongxin Wu, Rui Wang, Bo Yang, Hai Yu, Zhiliang Zhu, and Shing-Chi Cheung. 2018. Do the dependency conflicts in my project matter?. In *Proceedings of the 2018 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 319–330.

[65] wikipedia. 2023. *Unlicense, a public domain equivalent license for software which provides a public domain waiver with a fall-back public-domain-like license, similar to the CC Zero for cultural works. It includes language used in earlier software projects.* Retrieved March 21, 2023 from https://en.wikipedia.org/wiki/Unlicense

[66] Susheng Wu, Wenyan Song, Kaifeng Huang, Bihuan Chen, and Xin Peng. 2024. Identifying Affected Libraries and Their Ecosystems for Open Source Software Vulnerabilities. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[67] Y. Wu, Y. Manabe, T. Kanda, D. M German, and K. Inoue. 2015. A method to detect license inconsistencies in large-scale open source projects. In *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories*. 324–333.

[68] Y. Wu, Y. Manabe, T. Kanda, D. M German, and K. Inoue. 2017. Analysis of license inconsistency in large collections of open source projects. *Empirical Software Engineering* 22, 3 (2017), 1194–1222.

[69] Congying Xu, Bihuan Chen, Chenhao Lu, Kaifeng Huang, Xin Peng, and Yang Liu. 2022. Tracking patches for open source software vulnerabilities. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 860–871.

[70] 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.

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

[72] A. Zaimi, A. Ampatzoglou, N. Triantafyllidou, A. Chatzigeorgiou, A. Mavridis, T. Chaikalis, I. Deligiannis, P. Sfetsos, and I. Stamelos. 2015. An empirical study on the reuse of third-party libraries in open-source software development. In *Proceedings of the 7th Balkan Conference on Informatics Conference*. 1–8.

[73] S. Zhou, B. Vasilescu, and C. Kästner. 2020. How has forking changed in the last 20 years? a study of hard forks on github. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering*. 445–456.