Blockchain For Open-Source Software License Compatibility

Kypros Iacovou

University of Cyprus

Department of Computer Science

2025

Summary

This thesis explores a new method to tackle the complex issue of making sure open-source software licenses work well together, using blockchain technology. Open Source Software is everywhere, and it is important for developers and companies to follow the rules set by these licenses. We have designed, implemented and tested a web portal that uses blockchain to make it easier and automatic to follow these license rules.

The main point of such a system is smart contracts on a blockchain. These are like immutable agreements that help in managing and checking the licenses when someone downloads or uploads software. They are also very useful if someone uses existing software to build new software. The blockchain makes sure that everything is transparent and secure, so everyone can follow the license rules.

We start by looking at different types of open-source software licenses to understand their various rules and constraints. We also dive deep into blockchain technology, focusing on how smart contracts work, which is the most important concept of our solution. We also look at ways that we currently face this problem and why they might not be good enough, showing why our blockchain approach is better.

In building the system, we have created a web portal where users can create accounts, look for software, download it, and share their own software. We will later explain how we use blockchain and smart contracts to make sure the portal automatically checks if the software licenses are being followed correctly.

We will also describe the challenges we faced while creating and setting up the system, especially the technical concepts of using blockchain in a web portal. We tested our system thoroughly to make sure it works well to find that it vastly improves how open-source software licenses are managed and followed.

To sum up, this thesis shows that blockchain can be an amazing tool for solving an underrated problem in software development. It offers a new way to handle license rules more easily and securely. This thesis can be extended with more work in using blockchain for legal and regulatory needs in software engineering in general.

Commented [GK1]: I think better use throughout the text: Open Source Software

Table of Contents

Commented [GK2]: Should be in a separate page

Chapter 1 - Introduction

1.1 Background

Open Source Software and Licensing

Computers rely on source code, which programmers write, to execute tasks, while the system translates this code into a program. Accessing the source code is essential for developers to understand, change or improve the software. Open-source software ensures that this code is freely available, enabling transparency, collaboration, and continuous improvement. In this context, open-source code is not the absolute goal but rather a quite necessary condition for achieving software freedom which is the ability to study, modify, and distribute software without any restrictions. [3]

Commented [GK3]: Be careful not to use text from other sources to avoid plagiarism.
I run turnitin though on the whole text and similarity is on 7% which is okay but can be slightly improved.

Open-source software costs less to develop, license, and support than its closed alternatives. This cost-effectiveness makes it an attractive option for many organizations. Another advantage is that open-source software is often more secure and reliable, as it is constantly reviewed and improved by a large community of developers. The collaborative nature of Open Source Software also enhances performance, scalability, and stability, as experienced users refine and optimize the code over time. Using open-source software addresses many issues of software copyright violations. Furthermore, Open Source Software has the right licensing model from a corporate perspective. [6]

Commented [GK4]: Add the abbreviation the first time you refer to open source software, so that you can later use OSS directly in the text.

For the regulation of Open Source Software in terms of distribution, modification and usage there are several licenses. The main goal of open-source licensing is to prevent anyone from having full control over a work or project. A user of open-source software has much more freedom compared to traditional software users. They can distribute copies of the software (free or priced) due to the principle of "open distribution". Just like that, this principle lets us change the software and distribute our modified versions subject to the same flexible terms. According to Andrew M. St. Laurent [5], the one major requirement put forth in open-source

licenses is that distributed copies, whether original or modified, must be covered by a license that is compatible with the original.

Open-source software licenses generally fall into two categories: copyleft (reciprocal) licenses and permissive licenses. Copyleft licenses—such as the GNU General Public License (GPL) and Affero GPL (AGPL)—require that if any work is modified or derivative incorporating the original code must also be released under the same open-source terms. This enables publicly accessible improvements and prevents proprietary versions from being created without sharing modifications. This restriction on the other hand can limit businesses that are trying to develop closed-source products based on open-source code. In contrast, permissive licenses, such as the MIT License, Apache License 2.0, and BSD License, have minimal restrictions. They allow developers to modify, use, and distribute the software freely, including incorporating it into proprietary projects. These licenses do not impose legal obligations to be open, but rather use community and economic incentives to encourage contributions to the original project. A major difference between the two is permissive-licensed code can be incorporated into copyleft-licensed projects, but that is not always permitted the other way around, due to copyleft's stricter sharing requirements. This distinction influences how open-source communities collaborate and how businesses integrate open-source software into their products [1]

License Compatibility

License compatibility refers to the ability of different open-source software licenses to coexist and be combined in a single project without legal conflicts.

Because Open Source Software is created on the philosophy of code reuse and sharing, developers often feel free to combine software with different licenses. Yet, some licenses come with restrictions on the source code, which can make this incorporation difficult. For instance, permissive licenses, such as MIT or Apache 2.0 permit their code to be taken into a project under virtually any license, including proprietary licenses.

On the contrary, licenses like GPL or AGPL allow derivative works to be only used with more permissive or proprietary licenses, which becomes conflicting with these.

The compatibility of software licenses is important for developers, businesses, and the open-source software community because it determines whether components may be linked legally.

Illegal use can terminate project initiation and re-licensing is expensive after the incompatibility. Ensuring compatibility helps developers avoid legal disputes, maintain compliance, and foster innovation by permitting seamless collaboration across Open Source Software open source projects. Open-source software licensing enables collaboration and code reuse, but differences in license terms can create significant compatibility challenges. When licenses have contradicting requirements on how software is modified, distributed, or integrated, these issues arise. When licenses are incompatible, it may lead to legal disputes; and it prevents the combination of valuable open-source components. It is important for developers, businesses, and organizations using Open Source Software to understand these issues.

One of the most common compatibility issues occurs between permissive licenses and copyleft licenses. A GPL-licensed component, for example, cannot be legally integrated into an MIT-licensed project unless the entire project is re-licensed under the GPL. This can create difficult challenges for developers who wish to combine Open Source Software components with different licensing models, particularly when they intend to release proprietary versions of their software. Another significant issue arises from the distinction between strong copyleft and weak copyleft licenses. Strong copyleft licenses, such as the GPL and AGPL, require that all derivative works be licensed under the same terms. This means that any software incorporating GPL-licensed components must also be made open source under the GPL. In contrast, weak copyleft licenses, such as the Lesser General Public License (LGPL) and the Mozilla Public License (MPL), allow some flexibility.

Cloud-based software development introduces additional compatibility challenges, particularly with AGPL-licensed software. While the standard GPL applies only when software is distributed, the AGPL extends this requirement to software accessed over a network. This means that companies using AGPL-licensed software in web services or cloud applications must provide the complete source code, even if they do not distribute the software directly. This provision can deter businesses from using AGPL-licensed software in commercial cloud services, as it may require them to make open-source proprietary modifications.

Another compatibility issue arises from multi-license software, where a project is released under multiple licenses to offer flexibility. Whenever software is applicable to multiple licenses, the party who applies the license on this software chooses a license based on their needs. While it provides flexibility, it also creates complication due to compatibility when using it with other projects of a different license.

Finally, conflicts between Open Source Software licenses and proprietary software licenses pose significant challenges for businesses. Many Open Source Software licenses, particularly GPL and AGPL, do not allow integration with proprietary software unless the entire project is released under the same open-source license. This restriction can create legal and financial risks for companies that incorporate Open Source Software into their products without fully understanding the licensing implications.

In conclusion, Open Source Software license compatibility issues stem from differences in legal obligations, distribution requirements, and copyleft enforcement. Developers must carefully assess the licenses of all components they use to ensure that they can be legally integrated without violating licensing terms. As open-source adoption continues to grow, addressing these compatibility challenges is essential to fostering collaboration, reducing legal uncertainty, and enabling sustainable Open Source Software development.

Blockchain Technology

Blockchain technology is associated with the cryptocurrency Bitcoin, but its underlying principles extend far over and above digital currency. A blockchain can best be described as a distributed ledger, which is a decentralized database that records transactions across Source Software a computer network. Unlike traditional databases which are managed by a central authority – like banks in a financial network, blockchains operate on the peer-to-peer (P2P) network. In other words, no single entity controls the records. One of the features of blockchain is decentralization. In a normal centralized system, data is stored on one single server and controlled from a central organization. Instead, one of the characteristics of blockchain is that it distributes the data across Source Software all the participants in the network. As every participant (or node) in the Blockchain has a copy of the ledger, not one party can change or

Commented [GK5]: You should add a new section here on blockchain, as this section is too long.

delete the previous transactions on their own. Immutability is another key feature of blockchain technology, meaning once a transaction gets added to the blockchain and validated by the network, it cannot get changed or deleted. Transactions are collected into blocks, and each subsequent block is cryptographically bound to the previous one, thus creating a chain. Altering any previous record will necessitate altering all subsequent blocks as well, which cannot happen as of now due to network consensus. Blockchain technology is reliable for many tasks beyond just cryptocurrencies as it can be used to build applications for smart contracts, supply chain management, and digital identity verification, among others. Blockchain technology offers a secure, transparent, and efficient method for recording transactions by removing intermediaries, enhancing data integrity, and lowering cost.[4]

The advantages of blockchain technology can facilitate and secure digital transactions through its unique features. It is safe, transparent, and it has smart contracts. These things make it so popular in many industries. One of blockchain's primary strengths is security. Centralized databases are vulnerable to hacking and data corruption. However, blockchain has cryptographic encryption and decentralized data storage that can prevent this. It is more difficult for hackers to reach and manipulate data within the blockchain system. Each transaction gets an entry on the distributed ledger and consensus efforts verify it. This makes it incredibly challenging for a hacker to alter or manipulate transactions. It cannot be changed as anything added to the blockchain is defined immutable. Besides being decentralized, it also provides immutable data. Another key advantage is transparency. Due to the public ledger system of blockchains, all transactions are verifiable by the participants of the network which removes the need for any middlemen and chances of fraud. Blockchain provides transparency which is important for different industries like finance, intellectual property, and others.

Smart contracts are a key element of blockchain technology. Smart contracts enforce contracts automatically without the need for third-party oversight, thus cutting down legal claims, costs of transactions and administrative delay. They are widely utilized in decentralized apps (DApps), digital identity validation, and automated financial transactions.

Ethereum ranks among one of the most influential among the various blockchain platforms. In 2015, Ethereum was launched, and it was the first blockchain to launch programmable smart contracts. Developers can use these contracts in their decentralized applications. Its Ethereum Virtual Machine (EVM) allows for flexible and secure execution of smart contracts, making it a preferred platform for decentralized finance (DeFi), non-fungible tokens (NFTs), and enterprise blockchain solutions.

In summary, blockchain's security, transparency, and automation through smart contracts have revolutionized digital transactions. Ethereum, as a leading blockchain platform, has played a crucial role in driving blockchain adoption by enabling decentralized applications and innovative financial models across various industries.

Open Source Software License Compliance with Blockchain

Dealing with open-source software (Open Source Software) licenses can be complicated, especially where components have different licenses. Dev and organizations face legal and operational troubles due to complications like incompatible licenses, misattribution, and disobeying copyleft obligations. Blockchain technology provides a decentralized, transparent, and automated solution to these problems so that Open Source Software licensing terms are effectively adhered to. One of the most fundamental advantages of blockchain in Open Source Software license management is the ability to act as a tamper-proof, distributed ledger for recording software licenses. The blockchain can host each Open Source Software component and its license, helping developers track software's usage, modification, and distribution over time, and in real-time. This reduces the risk of license misinterpretation or loss due to the centralization of repositories. Using a blockchain-based system, software contributors can be recognized automatically and modifications to open source software (Open Source Software) projects can be securely filed.

Blockchain's smart contract functionality provides an effective way to automate license verification and compliance enforcement. Smart contracts are self-executing agreements that enforce licensing terms based on predefined rules. When a developer integrates an Open Source Software component into a project, a smart contract can automatically verify license

Commented [GK6]: Maybe the next paragraphs can also ne placed in a separate section with title: OSS License compliance with blockchain

compatibility between different software components, ensure proper attribution to original developers and enforce copyleft obligations, such as requiring modifications to be made publicly available. For example, if a GPL-licensed component is incorporated into a project, the smart contract could check whether the entire project is licensed under a compatible open-source license. If a conflict is detected, such as an attempt to integrate a GPL-licensed module into a proprietary software project, the smart contract can either prevent the integration or issue a compliance warning. Software license management is now not only low costing but also efficient because of its automated license verification which decreases human error.

The Blockchain technology makes Open Source Software license usage verifiable and immutable, thereby enhancing transparency in software development. Developers, organizations, and regulators can trace software components to their source to ensure licensing conditions are met at every level of software distribution. This is especially useful in big projects where many different developers are creating and adding on to the same code. Besides transparency, legal risk related to Open Source Software licensing is significantly reduced by blockchain. Many organizations face intellectual property disputes or compliance violations due to poorly documented license use. With the ability to store and authenticate Open Source Software license information, blockchain ensures that companies can prove compliance and avoid penalties. By introducing blockchain smart contracts into Open Source Software license management, the implementation of software compliance can become problem-free, transparent and legally secure. Using this technology offers a proactive answer to the complications that arise when enforcing Open Source Software licenses.

1.2 Problem Statement

Traditional Open Source Software compliance methods typically involve manual audits and legal reviews, which are often inefficient [2]. Developers and enterprises often use multiple Open Source Software components in their projects under different licensing terms. However, the identification and enforcement of license compatibility remains a big issue with compliance tools being able to only detect violations and not prevent them. Because of this, incompatible software components may be redistributed in violation of their licensing terms, exposing parties to liability, non-compliance, and litigation risk. A license incompatibility

occurs when software components having conflicting licenses are combined. In a specific instance, a project may try to combine GPL (General Public License) licensed code into its MIT licensed project. This might unintentionally force all the software to become GPL due to this compatibility issue. Most developers may not be aware of these restrictions so they may repost or redistribute software with incompatible licensing violating the law unknowingly. Tools such as license scanners and central repositories, which are already available and recognized as Open Source Software compliance tools, can detect a violation but will not stop the unintended reuse of code before they hit the public.

In addition, there are Open Source Software projects that change more often, it is not easy to see what parts are from which license. A problem related to the licensing of Open Source Software code occurred as it becomes more and more excruciating to track. If organizations won't have an automated verification system, they will be likely to redistribute non-compliant software—they would expose themselves to Intellectual property disputes and legal penalties. This problem can be solved by using Blockchain technology since it provides a tamper-proof, decentralized and automated platform for tracking and enforcing Open Source Software license compatibility. This thesis proposes a system to distribute software on a blockchain through function-level hashing. The re-posting of software that has incompatible licenses will also be prevented in real-time if the end-user verifies with the Blockchain whether they are compatible before re-distribution. Also, the public ledger guarantees transparency for the software, that allows anyone to check from where the software originates and its license conditions.

Given the inadequacies of traditional Open Source Software compliance methods and the growing complexities of license compatibility, this research explores how blockchain and function-level hashing can create an automated, secure, and legally compliant Open Source Software distribution platform. By ensuring that only license-compatible software can be reposted, this system aims to reduce legal risks, prevent compliance violations, and enhance trust in open-source collaboration.

1.3 Objectives.

The main aim of the research is to develop a blockchain-based system that ensures compatible open-source software license that does not allow redistribution of software with incompatible licenses. The existing Open Source Software compliance tools only confirm violations after they happen. But they don't exactly prevent them from happening. This thesis suggests a system that automates compliance enforcement through function-level hashing and smart contracts. This can allow only a license-compatible structure to be posted for each software by the developers. It aims to develop a decentralized platform for users to upload, download, and enforce Open Source Software licensing compliance.

The usage of Ethereum and Hardhat is employed to implement the system as both of them provide a tamper-proof environment. The system employs smart contracts to automatically verify whether a particular license has been complied with. When software is downloaded by a user, a smart contract is created between the author of the software and the downloader which stores the agreed-upon license terms and function-level hashes of all files. These hashes serve as a digital fingerprint to make sure code reuse is verified. To enforce licensing compliance, the system checks if a user uploading a new project is using functions downloaded before. The hashing procedure at the function level scans the uploaded project and matches it with the hashes existing in the downloaded software of the user. If there is a match, our system checks if the license for the new project is compatible with the older software. If the licenses are compatible, the upload proceeds. If a conflict is found, it stops them from posting the software with an incompatible license, ensuring compliance with Open Source Software license obligations.

In short, the research performed in this thesis ensures security, transparency, and automation in Open Source Software compliance with the help of blockchain's decentralization and cryptographic security. The system ensures that applicable licenses are respected and that the software is not subjected to an incompatible license. The goal of this system is to automate the Open Source Software license check and enforcement so that Open Source Software license conflicts are eliminated, unauthorized software can't be redistributed, and it leads to a legally-compliant and trustworthy open-source ecosystem.

1.4 Thesis Organization

This section explains the structure of the thesis and discusses the content of each chapter of the thesis. Divided into six chapters, the thesis discusses various aspects including the problem statement, methodology, implementation and findings.

Chapter 2 studies the Open Source Software licensing and license compatibility issues and the current compliance mechanism used. It further studies the blockchain technology and its essential characteristics and applications in respect of software compliance.

The proposed system has been designed and developed, as described in Chapter 3. The system architecture, as well as the importance of blockchain and smart contracts in automating license compliance along with the tools and technologies is discussed. Moreover, it discusses the security aspects of the decentralized compliance mechanism.

The implementation of the system is discussed in Chapter 4 which includes the design and deployment of smart contracts, implementation of function level hashing and verification of Open Source Software license compatibility before reposting of software. It also discusses the problems faced in the course of development and the solutions adopted.

Chapter 5 discusses the testing and evaluation of the system. It elaborates on the testing methodology. It discusses the different test scenarios and their outcome. It shows whether the system is able to enforce Open Source Software licenses. The chapter also talks about performance, security, and limitations.

Chapter 6 wraps up the thesis by highlighting the important discoveries and contributions of the study. It talks about how the proposed system will help in Open Source Software license management and mentions future research possibilities like improving scalability, interoperating with existing Open Source Software repositories and extending license verification mechanisms.

This provides a step-by-step approach which help us to understand the process of identifying the problem, developing the solution, implementing it, and evaluation to arrive a solution which will help in managing the Open Source Software license compliance through blockchain.

Chapter 2 - Background and Literature Review

2.1 Open Source Software licenses

Open-source software (Open Source Software) licenses are legal agreements that allow users to use, modify and share software within certain conditions. Most traditional commercial software licenses are designed specifically to limit the rights of users and reserve all control over the software to the creator. Open-source software (Open Source Software) licenses are purposely designed to promote software that the public can access and improve upon. For all its freedom, open-source is legally enforceable and defined software use and sharing is restricted to what is specified.

Open Source Software licenses are legal documents that operate under copyright law. They use the same set of rights allowed to authors to define how others can use their work. With its terms, Open Source Software licenses derogate the default position of copyright which is “all rights reserved” to “some rights reserved”. This guarantees the software remains open-source while preserving the original creator’s intentions. If developers or organization fails to comply with the Open Source Software license terms, it can amount to copyright infringement.[2]

The two general kinds of Open Source Software licenses with different legal and ethical implications are a permissive license and a copyleft license. Some permissive licenses are MIT License, Apache 2.0 License, and BSD License. They impose very few restrictions on using the software. You can change, reuse and combine the software with any type of software with very few conditions beyond simply maintaining attribution. Permissive licenses are licenses that allow software to be used, modified and shared with few restrictions. Permissive licenses—like the MIT

License—are very popular with software developers, in part because they are easy to understand. Yet, they also run the ethical risk of letting entities privatize open source work without returning to the community.

On the contrary, copyleft licenses, such as the GNU General Public License (GPL), Affero GPL (AGPL) and Lesser GPL (LGPL), impose stricter terms to preserve software freedom. This means that if someone takes the software, modifies it, and distributes it, they have to do it in the same way. Copyleft poses legal strength due to the enforcement mechanism that ensures software remains open-source throughout its lifecycle even if it is altered or used in derivative forms. As far as ethics are concerned, it also fits with the principles of openness, fairness and community contribution. This latter concept protects against situations where developers end up profiting from open-source computer code without giving anything back. Nonetheless, copyleft's legal rigidity often creates tension with commercial software development, where proprietary interests may be at odds with the need to release derivative work.

When you have a project that contains various parts which come from different open-source licenses, it creates a lot of legal complication. Software that is licensed under a strong copyleft, such as the GNU General Public License (GPL), cannot be integrated with proprietary applications, as this will create obligations that will lead to the entire combination being released under the same open-source terms. Failure to comply with these obligations results in license violations. When software is re-distributed without checking the licensing of everything in the software, it becomes incompatible and Open Source Software presents major risks for the developers and organizations. To avoid unintended legal and operational consequences, it is important to properly evaluate all components for license compliance.

When exploring the ethical component of Open Source Software licensing, you will discover the principle of "freedom" for software, the users of the software system, and the community in general. While permissive licenses encourage flexibility, copyleft licenses instill a kind of digital commons that requires innovations developed with this Open Source Software to be made available. Keeping up compliance as well as the spirit of open-source software development can be accomplished by navigating through these legal and ethical frameworks.

With Open Source Software being developed by an individual or enterprise at a large scale, managing diverse licenses has become complex in a big project. As the complexity of managing

the various licenses has increased, there is now a greater requirement for automated tools and systems that can check license compatibility and avoid inadvertent infringement of the law. The next few sections will explore the existing studies on the issues of license compatibility and the technologies proposed for the same.

Open source software licenses are designed to control the way software can be used, modified, and re-distributed. The licenses in question differ widely in their legal specifications and intended usage scenarios, which means that the choice of license is a strategic decision that impacts the growth, adoption and legality of a project. Open Source Software licensing broadly fit into three broad categories: permissive licenses, strong copyleft licenses, and weak copyleft licenses. Different model has different focus areas i.e. increase of user adoption or sustainable software freedom, which makes a way to different industries.

Permissive licenses, like the MIT License, Apache License 2.0 and BSD License, have the simplest restrictions on the use of the software. Licenses of this kind allow developers and companies to use, modify and embed parts of open-source software into proprietary software without having to make the modified version open source. Due to their flexible nature, permissive licenses have become the mainstream choice for commercial software development, startup space and cloud-based offerings.

For example, React.js is one of the most notable JavaScript libraries. It falls under an MIT License which encourages further use in open-source and commercial web applications. In the same way, TensorFlow, which is a commonly-used machine-learning environment, makes use of the available Apache 2.0 license so that sectors such as artificial intelligence, healthcare technologies and financial services can develop proprietary solutions. In such sectors, permissive licenses facilitate the speedy development of their overall innovation while allowing organizations to choose how much of their product to share.

The strong copyleft licenses, are the GNU General Public License (GPL) and the Affero General Public License (AGPL). According to the GPL, if an entity modifies the GPL code and then distributes it, it must share the modified code under GPL as well. In projects where software freedom is paramount, and proprietary capture must be frustrated, these licenses are usually used like infrastructure software, operating systems, and digital rights tools.

The Linux kernel is a great example of strong copyleft licensing at work, preventing any proprietary capture while ensuring open-source modifications. The AGPL broadens these requirements to software made accessible via a network, rendering it a famed option for web platforms and decentralized apps. AGPL is a license that stops corporations from modifying or profiting behind privately off of community software. By enforcing this clause, it ensures that source code has to be shared.

The Lesser GPL (LGPL) and the Mozilla Public License (MPL) are weak copyleft licenses that are between open access and proprietary flexibility. It is Open Source Software with the help of these licenses to link a proprietary application to the libraries without open-sourcing the app itself. This means that often these are used in the case of software libraries, cryptography, middleware as it allows them to get distributed without any control over them. For example, FFmpeg, a library for transcoding multimedia files, mentions LGPL in its license tags to allow being used in open-source and closed-source projects. In the same way, this license is for modular projects, so only the files someone changes must be shared. The rest can remain proprietary.

In reality, the choice of license is based on the practical usage and strategic goal of the project. Permissive licenses are common in domains like AI, fintech, and web where speedy development, flexibility and commercial advantage are the main drivers. Projects that intend to secure long-term freedom of software and community-driven development favor strong copyleft licenses. Projects like these are mostly found in operating systems, blockchain technologies and privacy-oriented projects. Weak copyleft licenses are used where modularity and mixed-source models are more needed, especially in enterprise middleware, frameworks, and SaaS.

The appropriate licensing model determines how software can be used and shared, affects community and corporate involvement, and establishes the legal parameters of a project's development and distribution. This makes it a huge decision. Not aligning the licensing strategies to the project goals may cause legal risks, complicate adoption, or impose unanticipated obligations during redistribution or commercial deployment.

2.2 Blockchain Architecture and Key Features

Blockchain is a technique that creates a transparent record of data. Thus, it removes the need for a third party. Blockchain is a distributed ledger technology designed to record all transactions in a

safe and secure environment. The technology was first presented as an underlying mechanism for Bitcoin [7]. After that, it has grown into a versatile system that has uses well beyond bitcoins [18]. Today, these systems are increasingly used in the field of finance, supply chain management, healthcare, digital rights enforcement, and many other fields.

The figure below illustrates the basic workflow of blockchain technology. New data, such as a transaction or software license agreement, is recorded as a block. This block is then broadcast to all nodes in the decentralized network for validation. If the nodes reach consensus, the block is approved and added permanently to the chain. This process ensures data integrity, security, and transparency — all essential for Open Source Software license enforcement in our system.

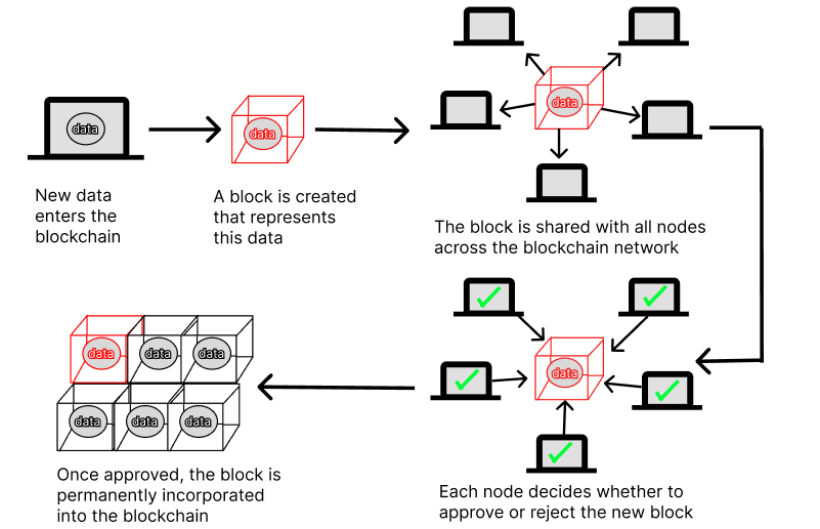


Figure 1: The Blockchain Process

The most basic blockchain architecture consists of blocks chained together one after another. To explain, a block contains a set of verified transactions in addition to a timestamp. The block will also contain the cryptographic hash to the previous block. As a result, the structure is sequential and immutable; if any block is changed, subsequent hashes must be recalculated, and this is hardly feasible in large decentralized networks. The design ensures that the data held cannot be tampered

Commented [GK7]: All figures need to be numbered. You created the figure again, correct? I hope you changed the text as well, it should not be the same.

Formatted: Centered

with. A fundamental feature of blockchain is decentralization. Blockchain networks do not depend on a central authority. So, consensus algorithms, such as Proof of Work (PoW) or Proof of Stake (PoS), verify transactions. They help ensure consistency Open Source Software distributed nodes as it removed the need for intermediaries and improved resilience from single points of failure.

Another critical property of Blockchain is immutability. Once the records are there on the blockchain, they are consensus and cannot be changed. This is why blockchain is valuable in cases where auditability and data are crucial in a financial system, healthcare records, and digital asset management.

Transparency is also inherent in blockchain design. Anyone who takes part of the network will always see how many transactions occurred at any instance in time. A feature of private or permissioned blockchains is that they control access but the auditing is verifiable.

Also, smart contracts are another feature of blockchain that allow people to create a contract that automatically executes so long as certain conditions are met. These agreements lessen the need for mediators and make systems such as financial services, supply chains and digital licenses automatic. In total, what you have is excellent for ensuring integrity and creating systems that can replace data-intensive exhaustive systems.

Smart contracts are self-executing contracts with the terms of the agreement directly written into lines of code. In the 1990s, the idea of smart contracts were conceptualized from Nick Szabo [8]. Today, smart contracts are a key component of many blockchain platforms, including Ethereum, which allow for complex digital interactions to take place in a secure, transparent, and trustless environment.

Smart contracts provide fundamentally useful mechanisms for automating processes that would otherwise require manual intervention or third-party enforcement. Smart contracts are self-executing agreements in which the terms are coded into the agreement itself. By ensuring that all parties adhere to the predefined conditions without ambiguity, this automation reduces administrative overhead, eliminates the potential for human error, and minimizes the risk of disputes. In terms of compliance, smart contracts could improve the enforcement of rules, regulations and contracts. Smart contracts offer a transparent and accountable audit trail since all contract execution and transactions are recorded in the blockchain in an immutable manner. In

industries where rules have to be ensured like financial services, supply chain, healthcare, and intellectual property, smart contracts can automate the verification of conditions to ensure compliance.

Smart contracts can be programmed to ensure license terms in digital rights management or open-source software licensing automatically. They can verify usage rights, ensure compatible licensing terms on reuse of software components, and block unauthorized redistribution. Furthermore, they lessen legal risks by putting compliance norms directly into the transaction logic. They also allow for complex licensing conditions to be enforced without the need for manual audits or legal interventions. Moreover, they boost trust in decentralized settings where parties do not know or trust one another. Due to the deterministic nature of contract execution, which is reliant on blockchain consensus, no party has to depend on a third entity.

All in all, smart contracts help decentralization through automation. They are advantageous because they are also legally binding. Since smart contracts are self-executing, verification of condition is deterministic, governed by the consensus of the blockchain, and all parties can rely on them.

2.3 Real-World Cases of Open Source Software License Conflicts

Open-source software(Open Source Software) development must take into account license compatibility meaning that mixing and matching components licensed under different licenses can lead to legal problems and business risks. Real-life examples show how this incompatibility has caused conflicts, compelled projects to be redesigned or led to lawsuits. These cases demonstrate how serious Open Source Software licensing issues are in the real world.

One of the most clear examples is the Oracle-Google fight in which oracle sued google for using its Java API in Android operating system. Google made Android, which was based on Java, but instead of getting a commercial license from Sun (later Oracle), it reimplemented certain Java APIs. Oracle thought that Google was copyrighting a portion of Java that was not open source due to its integration of APIs. The matter proceeded to the United States Supreme Court. The Supreme Court made a ruling in 2021 that APIs were used under fair use without allowing Oracle upping their work for free. Yet, the case demonstrates the legal uncertainty that exists when it comes to

Commented [GK8]: Is there a reference for this? Even from an online site. You should add it.

Commented [GK9]: ? Without?

Open Source Software license and proprietary software integration; particularly when the same piece of code is involved, there are two separate licensing terms.

The well-known Ghostscript project is another famous example. This refers to an open-source PostScript interpreter widely in use. The open-source Ghostscript program often caused legal issues when integrated into proprietary software. Multiple companies have infracted the GPL of Ghostscript by embedding it into their own proprietary software, simply because the license is permissive. This resulted to filing of lawsuit by Artifex Software, the maintainers of Ghostscript, including a prominent lawsuit against South Korean company Hancom. The court of the U.S.A. ruled the breach of the terms of the GPL- law is enforceable as a breach of contract, thereby making the copyleft licenses legally enforceable. Businesses that don't comply with the licenses are at risk.

A further example relates to the broader incompatibility between GPLv2 and Apache License 2.0. While both are open-source licenses, they have different terms and cannot be used together. In particular, the patent terms in particular do not agree. The GPLv2 does not state patent protection while Apache 2 does include a grant. Because of this, projects that merge components with GPLv2 and those with Apache 2.0 run into problems. As early as Android, Google made an effort to avoid intermixing the two licenses to not cause downstream legal issues.

The above cases demonstrate that incompatibility of licenses is not just a theoretical phenomenon as it results in costly consequences in practice. Open-source software has challenged software design and business model issues that are much more than a dispute in the court of law over licenses. As software systems grow more complex, integrating components governed by different licenses increases the probability of legal conflicts and commercial risks [3]. Lawsuits against the Ghostscript project show that violations of the GPL are enforceable, and violators can face a successful lawsuit along with penalties.

License incompatibility may restrict software architecture and design. Developers may choose not to use certain Open Source Software libraries in their proprietary projects to avoid triggering copyleft obligations, increasing their development overhead or rework. License incompatibility can impact business sector as well. Dependence on proprietary software distribution may render companies' business models incompatible with copyleft restrictions. Firms face lawsuits, reputational costs, and investor concerns from violations. Operational as well as legal resources

are used to manage the compliance in many sectors like finance and healthcare with strong oversight.

In total, the incompatibility of licenses entails legal, technical and business risks. Some things that can help limit these risks are compliance programs, smart license reviews and technologies that can automate the license validation.

2.4 Related Work

The management of open-source software (Open Source Software) licenses has long been a problem for software developers, especially in bigger projects. Most research so far has been concerned with building tools and frameworks to help developers identify license types, detect conflicts, and ensure compliance during integration and distribution.

Conventional Open Source Software license management relies on the support of automated scanner tools such as FOSSology and Black Duck which help to scan the codebase of software, identify the licensed embedded in it, and flag incompatibility issues. Tools of this kind help an organization identify issues of non-compliance, but generally operate as reactive mechanisms. They help identify conflicts after the development phase has occurred which leaves room for human error and increases legal risk. Also, when the system is large, modular and diverse, verification of licensing at the function level is rarely possible.

Most models based on blockchain are still conceptual, or proof-of-concept-based. Most existing solutions cannot work at a function-level granularity, which is important for accurate tracking of code reuse and enforcement of complex license terms when multiple software components get mixed together. Also, past studies seem to ignore the automated enforcement of compatibility but focus more on license recording and visibility.

This shows the need for a system, that will not only record this data but also stop incompatible software being redistributed. Through the integration of function-level hashing and the use of automated smart contract enforcement, the system proposed in this study will go beyond previous models and offers live license compatibility checks with live enforcement. As compared to other

existing solutions, this one offers a broader coverage of legal, technical and business challenges faced in Open Source Software license management.

Current literature on Open Source Software license management and the use of blockchain for enforcement has gaps in automated, function-level license enforcement and need for further research. The next chapter details the efforts made to design and implement a blockchain-based system that ensures real-time Open Source Software license compliance.

Chapter 3 - Methodology

3.1 System Overview

The system built in this study is a blockchain-based Open Source Software license compliance system that applies automated function-level verification on Open Source Software programs. The core of the architecture will utilize smart contracts, function-level hashing and decentralized storage of license information mechanism to prevent license incompatibilities arising in redistribution of software. The system provides a transparent automated compliance mechanism to prevent the occurrence of software with conflicting licenses being redistributed.

The platform provides a hosted web interface that allows registered users to interact with software projects on a local server environment. Once users sign in, they will be able to upload a new software project or download any existing software project. A smart contract is deployed over Ethereum each time the user downloads the project. This contract states the license agreement between the author and downloader and creates all function hashes of the software downloaded. These hashes uniquely identify the code at the function level. This allows the management system to identify whether such functions were previously used in any future upload.

The upload process of the system contains a verification. When a user uploads a new software, the system scans the uploaded files, generates function-level hashes, and compares these hashes with the stored hashes of all previously downloaded and user-associated projects. If the system sees equivalence, it obtains the license terms linked to the original function and runs an automatic compatibility check against the license declared for the new upload. If the licenses are compatible, the project is successfully uploaded on the platform. The upload will stop if there is a license clash;

for instance, if someone tries to relicense a GPL code under a non-compatible license, then the user will be notified.

This architecture avoids unintentional or malicious license violations and also maintains a permanent record of licensing agreements and code reuse. The system lessens the legal and administrative burden of Open Source Software license compliance by automating checks and evidence storage on the blockchain while providing transparent and verifiable compliance history for users and projects.

3.2 Tools Technologies and Languages

Using blockchain technology, web development technology, and server-side programming we developed a system that automates Open Source Software license compliance. Ethereum blockchain is the central point of the system as it can utilize their strong smart contract functionality and support for decentralized application development.

The Hardhat environment is used to ease writing, testing and deploying of smart contracts on the Ethereum network. Hardhat offers a versatile framework that eases the use of blockchain and allows for local testing of contract behavior before deployment.

The system's smart contracts are written in Solidity which is the main programming language for contracts on Ethereum. By using Solidity, we can encode license agreements, store function-level hashes, and automate license verification processes on the blockchain. The smart contracts are created to act autonomously, meaning once they are deployed they can manage the terms of the license and verify compliance without external action.

Commented [GK10]: You can add later in the text an example of a smart contract of the system in solidity, unless you have done this already.

JavaScript, PHP, and CSS are the main components used in its front-end and back-end. JavaScript manages activities on the client side, user actions, and interaction with the deployed smart contracts. The backend created in PHP is used for user authentication, project management, and database operations. PHP scripts help the user interface (UI) interact with MySQL. MySQL will be used to store the user profiles of the customers, project's metadata and blockchain transaction reference. The entire system runs on a local XAMPP server that aids in hosting, database interaction, and all server-side processes for development.

Function-level hashing is fundamental for the system's compliance mechanism. A cryptographic hashing algorithm like SHA-256 is used to hash, or recognize, each function in a software project or system. Through this method, the system can identify code reuse at a function level, even if file names or the formatting have changed. By emphasizing the analysis of functions rather than entire files, the system improves accuracy in license tracking and decreases occurrence of false positives in license conflict detection.

All these technologies combined build a working prototype which can automate Open Source Software license compliance. The blockchain technology is transparent and immutable while other tools for the web will ensure accessibility and usability within the local environment. These components form the technical basis required to attain license verification and enforcement reliably and effectively.

3.3 Smart Contracts Design and Workflow

The smart contracts in this system shape automated license tracking and compliance verification which is the main mechanism. Each smart contract will record essential data, such as the agreement on the license conditions and the function-level structure of the downloaded project, for every download. The Ethereum blockchain's security and transparency make them ideal for contracts involving license tracking and compliance verification. They are designed to store and preserve the data of an open-source download, including the license terms and the function-level structure of the project that was downloaded.

Commented [GK11]: You should indicate exactly what each cntract includes, maybe you can use a list for this.

When you download a project, a smart contract gets deployed on Ethereum automatically. This particular contract stores the license under which the program is downloaded, clarifying the conditions for the downloader. Along with the license, the smart contract records a hash for each function in the project that has been downloaded. The smart contract also records a unique hash for every function in the project that is downloaded. These function level hashes allow the system to identify reused code exactly and not allow such reuse.

The smart contract is activated once more to ensure compliance during the upload process. When the user tries to upload a new project, the hashes of each of the functions in the submission are generated and compared with the hashes of the smart contracts that the user has previously created. If there are no matches, you can upload freely. If the system finds one or more functions match

code from a previous download, it automatically retrieves the original license back from the smart contract. Next, a check for license compatibility is performed to see if the license of the new project is legally compatible with the license of the reused code. After the verification of compatibility, the upload is permitted. Otherwise, the upload gets blocked by the system along with a notification on the conflict to the user.

Commented [GK12]: Explain at some point later exactly how the license compatibility check is done, with the use of a table (unless you have done this already).

Commented [GK13]: For this you can give later in the text an example of how a notification shown to the user looks like. Maybe you can use an example use case of the system (unless you have also done that already).

The design verifies that compliance is made proactively to stop the redistribution of code under incompatible licenses. The system removes the need for legal examination or post-distribution audit by putting the license logic in the smart contract. As soon as something is uploaded, the blockchain can make sure that an obligation is being followed.

In the current version of the prototype, wallet addresses still have to be manually managed. The platforms on blockchain usually provide automatic wallet creation and integration of wallets. However, the prototype of the system requires that the wallet addresses of newly registered users are manually entered in a configuration file by the system administrator. While it works for development, it cannot be scaled, nor can users add addresses independently. Future versions of the system could automate the generation and management of wallets, so that they increase usability and allow scalability without administrative actions.

Even though there is this limitation, the smart contract workflow serves as a workable example of how the use of blockchain technology can automate Open Source Software license compliance at the functional level. It is all verifiable, legally compliant, and enforced without a person's oversight.

3.4 License Verification and Function Hashing Method

A key element of the system's approach is to use function-level hashing to effectively identify code reuse between software. This mechanism allows the functioning of system not merely on file-level or project-level comparison, but also on individual functions that are the logic unit of code. This level of detail is needed so that the system can track even partial reusage of code and validate their licenses.

When a user downloads a software project, the system processes the project and extracts each function. For each of the functions identified, a cryptographic hash is created using a fixed hashing algorithm such as SHA-256. The hash is being generated for a function which represents it

completely as far as its content and structure is concerned. Subsequently, these hashes, as well as the license under which the project was downloaded, are permanently stored inside the smart contract.

Commented [GK14]: What if a function is too short though? E.g. only 1 line of code, just for printing purposes, it shpuld be okay to reuse such functions without a problem. Maybe you can mention that future work can allow the original uploader of the software to indicate functions that are too generic to be under license restrictions or functions that are placed under the public domain (and hence also do not have license restrictions and can be freely reused).

While uploading, the same extraction and hashing of the new project's functions is performed. The hash is compared to set of already stored smart contract hashes of the user from downloads done earlier. If there are no matching hashes, it means the project has not reused any code that has been downloaded before and the upload can proceed.

When a hash matches, it means that the user is attempting to reuse a function that has been downloaded before. The license compatibility logic is activated at this point. The system first retrieves from the blockchain record the original license of matched function and compares it with license declared for new upload. The compatibility check follows predefined protocols based on familiar Open Source Software license interactions. For instance, a piece of code that is licensed under GPL cannot legally be relicensed under a permissive license such as MIT because doing so would violate the GPL's copyleft requirements.

If the software determines the license of the new project is compatible with the license of the original project, the upload is approved. If the system detects that the license of the new project could not fulfill the obligations of the reuse project's license, the upload is automatically rejected and you are notified of the incompatibility. The automated mechanism guarantees that Open Source Software license obligations are fulfilled, in a real-time manner. Using hashes at the function level allows this system to offer more accurate results than licensed scanners that function at the file level and so reduce false positives and omissions. In addition, the software's license compatibility logic guarantees compliance with legal constraints, sparing users from the need to decipher complex licensing language and drastically reducing the chances of mishaps due to incompatibility.

3.5 Security and Compliance Considerations.

The system development priority is to automate Open Source Software license validation and ensure the integrity of all transactions and license agreements from tampering and alteration. One major security benefit of our system is the fact that blockchain records are immutable. When a smart contract is deployed on Ethereum, the license terms and function-level hashes from the

downloaded project are permanently stored on the blockchain. The implication is that agreements cannot be erased or modified after a transaction is recorded. Hence, this allows no dispute whether the licensing terms of the downloaded project were followed or not. Since blockchain data cannot be manipulated, the case of what was downloaded will always be clear. Thus, legal obligations can be expertly enforced via smart contracts.

Nonetheless, the present implementation has a few limitations inherent to prototypes that may present risks. One limitation like this is the manual processing of wallet addresses. A system administrator must register the wallet of any user in the configuration file for the user to be able to operate with the system. Although it functions in a development setting, it is potentially dangerous if mishandled. A scalable way to do this would be to automate wallet generation and allow users to self-register. This will minimize admin effort and reduce reliance on a system admin.

The accuracy of function-level hashing as a tracker of code reuse is another security issue. While this hashing mechanism is very useful to identify identical functions found in other projects, it does not detect situations in which the code is altered very slightly but does the same job. Future improvements could research smarter analysis or the usage of machine learning to detect code similarity for enhanced strength against simple techniques.

Although the system at the moment is dedicated entirely to Open Source Software, the architecture offers the probability of future expansion into proprietary software. The system will allow secure payment for proprietary software and full compliance with licensing obligations by incorporating payment mechanisms and extending smart contracts for licensing fees. This ability is outside the scope of this research and is considered a direction for future work.

Via this thesis implementation, it is successfully demonstrated how enhancing blockchain technology can help with security and compliance with Open Source Software license. With the help of immutability, automation and decentralized verification, it reduces reliance on manual compliance checks, mitigating legal risks and ensuring that software is distributed in accordance with licensing requirements.

Chapter 4 - System Implementation and Discussion

4.1 System Architecture

The suggested system is a platform which has a combination of blockchain technology, server-side processing, and client-side functionality to automate open-source software license compliance checks. The architectural design includes a modular structure so that the concerns of the frontend user interface, the backend processing, as well as the blockchain layer responsible for immutability will be entirely separate.

The figure below (Figure 2) shows the architecture of the system. It illustrates the interaction between the user, frontend (JavaScript), backend (PHP and MySQL), and blockchain smart contracts. The user can upload or download projects via the frontend interface. On upload, the frontend extracts function-level hashes and interacts with the backend for user/project management and license compatibility checks. Simultaneously, it communicates with the LicenseManager smart contract on the blockchain to verify and store license information. When downloading, a smart contract agreement is triggered via the DownloadAgreement contract to ensure license acceptance is recorded immutably on-chain.

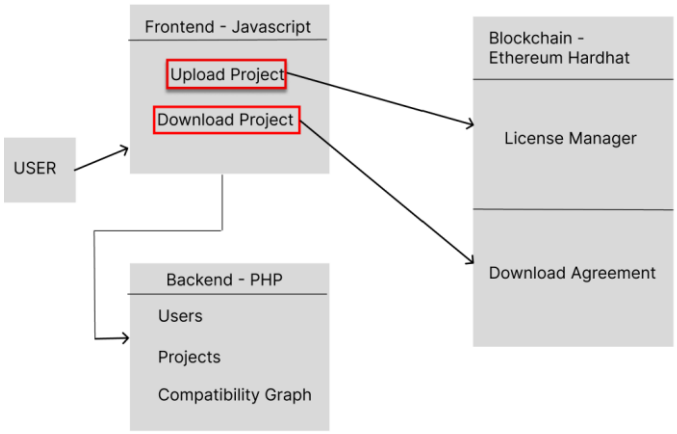


Figure 2: System Architecture Diagram

Formatted: Centered

The frontend layer is the main point of the system, which is mainly developed in JavaScript and executed in the user's browser. It handles user interactions, file uploads, and the initial processing of software projects. After a user submits a project, the frontend reads each of the source files and does a function-level extraction and hashing.

By applying the SHA-256 Algorithm, the JavaScript logic generates a distinct hash for every function. This implies that even if any code is reformatted, its segments can still be uniquely identified. This design also allows the system to detect partial reuse in later uploads of the same project.

The backend is created in PHP and runs on a local XAMPP server. It manages the user login, and handles the file storage. It also maintains the MySQL database of the system. The database saves project metadata, user profiles, and a structured representation of the rules regarding license compatibility, defining which Open Source Software licenses are legally compatible. When the compliance check is requested by the frontend, the backend checks that license graph to assess whether the intended license of the user is valid, given any detection of code reuse. In order to implement this database table with we used the following the following graph to create this mapping.

Commented [GK15]: Add the license graph from the prior publication and mention how it is mapped to a DB table for implementation purposes.

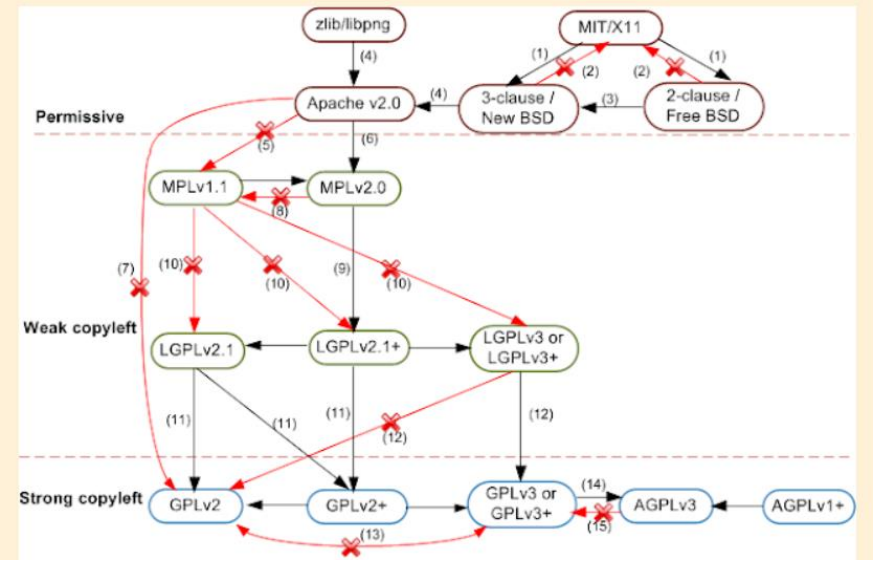


Figure 3: License compatibility graph for database table [19]

The Ethereum network is used to implement the blockchain layer with Hardhat development framework as well as Solidity smart contracts. This layer is formed by two main smart contracts, DownloadAgreement and LicenseManager contract. Together, these contracts take care of storing and managing the licensing agreements, recording the function-level hashes of downloaded and uploaded projects, and tracking the metadata of these projects. The blockchain ensures the immutability of license acceptance records and function hash logs, providing a transparent and verifiable history of each user's interactions with the platform.

The entire system workflow is sequential and integrated across these components. After the user downloads the project, the frontend initiates a blockchain transaction to record the license agreement. When a new project is uploaded, the system hashes its functions, and queries the blockchain to see if any function hashes that were previously downloaded match. If there is a match, the backend does a license compatibility check, and it retrieves the original license terms. The upload of the project is permitted only if the licenses are compatible. Otherwise, the system prevents the upload and informs the user about the conflict.

This design checks Open Source Software license compliance and enforces action before the upload. By merging client-side code analysis, server-side compatibility logic, and the blockchain's immutability, the system creates a strong, transparent, and legally sound environment for open-source software reuse management.

4.2 Smart Contract Design

The smart contracts developed for this system are the basis of the platform's automated license tracking and compliance mechanism. The contracts are coded in Solidity and deployed on the Ethereum blockchain using Hardhat framework. There are two main functions. First, to record the licensing agreements during downloads. Second, to verify and store the project function hashes during uploads.

DownloadAgreement is the first contract, which records the acceptance of the license when the user downloads a software project. When someone downloads the software, this agreement stores the wallet address of the downloader, the unique ID of the project, the type of license, and the timestamp of the download. This permanent record uses the legal agreement under which the

Commented [GK16]: You can given an example here for a smart contract.

software was acquired. The data is stored on-chain where it can be retrieved for verification in the future.

To demonstrate this functionality, the `acceptLicense()` function from the DownloadAgreement contract is shown below. This function captures the legal agreement established during a software download and stores it immutably on the blockchain.

```
function acceptLicense(uint256 originalProjectId, string memory licenseType) public {
    licenseRecords[originalProjectId].push(
        LicenseAgreement(msg.sender, originalProjectId, licenseType, block.timestamp)
    );
    emit LicenseAccepted(msg.sender, originalProjectId, licenseType, block.timestamp);
}
```

Formatted: Font: (Default) Courier New, 11 pt

Commented [GK17]: Better use this font (Courier New) for source code,

Formatted: Font: (Default) Courier New, 11 pt

This function records the wallet address of the downloader (msg.sender), the ID of the downloaded project, the selected license type, and the exact time of agreement using block.timestamp. These details are pushed into a license record array on-chain, allowing for reliable retrieval at any point in the future.

In addition to storing the agreement, the function emits an event (LicenseAccepted), enabling the frontend interface to respond in real time by, for example, initiating the project download or confirming successful license acceptance.

Although the logic is relatively simple, it plays a critical role in the system's compliance infrastructure. The record created by this function serves as an immutable indicator of user consent to specific license terms. This becomes essential when that same user later attempts to upload a derived project: the smart contract can then check whether the license selected for the new upload is compatible with what was originally accepted. Without this download-time record, license tracking would depend solely on backend logic, which lacks the transparency and permanence of blockchain storage.

On the other hand, the second contract is LicenseManager, which is more complicated. It manages the uploads, tracks hashes of functions, and enforces license rules. When a user uploads a new

software project, this contract will register the metadata of that project including the address of the uploader, the ID of the project, any parent project it may come from, and the license selected. More importantly, the contract stores an array of function-level hashes from the uploading project. The project's functional logic is signified by these hashes helping with the precise detection of code reuse on the platform.

Commented [GK18]: Same comment here for an example.

A key aspect of the LicenseManager contract is its straightforward license compatibility verification during uploads. If a project gets uploaded, which claims to be derived from a project that was downloaded before, the contract scans its database to see if the new license is compatible with the license under which the parent project was downloaded. If no license is compatible, the project is flagged on-chain, preventing silent non-compliance. The backend PHP logic that queries a license graph that exists in the database ensures further license compatibility checking after the early checks.

To illustrate this logic, the following Solidity function from the LicenseManager contract demonstrates how uploads are handled and license checks are enforced:

```
function uploadProject(
    uint256 projectId,
    uint256 parentProjectId,
    string memory licenseType,
    string[] memory functionHashes
) public {
    bool isFlagged = false;
    if (parentProjectId != 0) {
        bool isCompatible = false;
        LicenseAgreement[] memory agreements = licenseRecords[parentProjectId];
        for (uint256 i = 0; i < agreements.length; i++) {
            if (
                keccak256(abi.encodePacked(agreements[i].licenseType)) ==
                keccak256(abi.encodePacked(licenseType))
```

```
            ) {
                isCompatible = true;
                break;
            }
        }
        if (!isCompatible) {
            isFlagged = true;
        }
    }
    uploadedProjects[projectId] = UploadedProject(
        msg.sender,
        projectId,
        parentProjectId,
        licenseType,
        isFlagged
    );
    projectFunctionHashes[projectId] = functionHashes;
    emit ProjectUploaded(msg.sender, projectId, parentProjectId,
licenseType, isFlagged, functionHashes);
}
```

This function plays a central role in the license enforcement workflow. It first checks whether the upload references a parent project. If it does, the contract compares the current license with licenses accepted by the uploader for that parent. If the licenses do not match, the new upload is flagged on-chain. This mechanism ensures that projects cannot silently incorporate incompatible code. The function also stores the uploaded function hashes, making reuse detectable in future uploads.

Similarly, when a project is downloaded, the platform uses the following function to store the user's acceptance of the license terms:

```
function acceptLicense(uint256 originalProjectId, string memory
licenseType) public {
```

```
    licenseRecords[originalProjectId].push(
        LicenseAgreement(msg.sender, originalProjectId, licenseType,
block.timestamp)
    );
    emit LicenseAccepted(msg.sender, originalProjectId, licenseType,
block.timestamp);
}
```

This function records a legally significant agreement between the downloader and the software license. It creates a timestamped record of who downloaded which project under which license. These records are later used to determine whether future uploads are legally permissible under the same or compatible licenses.

Both smart contracts trigger events to log actions such as license acceptance and project uploads. These events are used by the frontend application to provide real-time feedback and support asynchronous communication between the client interface, the blockchain, and the server-side components.

The project uses blockchain features, such as immutability, transparency, and decentralization to securely store license agreements, project metadata, and function hashes. This ensures that once data is recorded on-chain, it cannot be modified, offering a verifiable compliance history for each project and user. Furthermore, by placing key compliance logic directly in the smart contracts, the system reduces reliance on external enforcement and protects against both accidental and intentional license violations.

Both smart contracts trigger events to log actions such as license acceptance and project uploads. The frontend application can give real-time feedback to users and make asynchronous operations possible with the blockchain and server components.

The project makes use of the blockchain's features of immutability, transparency and decentralization by inserting license agreements, project metadata and function hashes in smart contracts. This design guarantees that after a licensing record or a function hash is placed, it cannot be modified, assuring a verifiable compliance history that cannot be tampered for each project and user. In addition, the key compliance checks will be inserted at the blockchain level through smart

contracts which will reduce reliance on external enforcement mechanisms. This will protect the system from accidental or intentional license violations.

4.3 Frontend and Blockchain Integration

Integrating the frontend interface, the backend logic, and the blockchain smart contracts is crucial for the system's execution. All actions the user can do related to the system like downloading the project, uploading new software or getting license compliance checks are consistently tracked and validated across all layers of the system due to this integration.

We are using JavaScript to build the frontend, where the user interaction and processing is done. When a user uploads a project, the frontend extracts the function definitions from the source files and generates cryptographic hashes for each function using the SHA-256 algorithm. This is done completely on the client side.

Upon generating function hashes, the frontend communicates with the Ethereum smart contracts. By using ethers.js, the application connects to the LicenseManager smart contract to retrieve the function hashes stored during earlier downloads. The frontend compares the new upload's hashes to those stored on-chain. If a match is found, the system detects potential code reuse and initiates license compatibility verification process.

The blockchain and the backend work together to assess license compatibility. After finding matching function hashes, the frontend uses the `getProjectDetails` function of the LicenseManager contract to retrieve the corresponding license information from the blockchain. The backend PHP script then receives this info and queries a license compatibility graph from the MySQL database. The system uses the compatibility graph, which defines the known legal compatibility rules between licenses. This allows it to determine whether the license the user selected for the new upload is permitted, given the license of the reused code.

Formatted: Font: (Default) Courier New

When validated successfully, the frontend calls the `uploadProject` function of the LicenseManager smart contract. The transaction records relevant metadata, license, parent project, and all function hashes of the blockchain. As a result, the upload is verified to be legally compliant and immutably logged.

Formatted: Font: (Default) Courier New

To download the project, the frontend initiates a license agreement, by interacting with the DownloadAgreement smart contract. The wallet address, the project ID, and the terms of the license on-chain are recorded on-chain, and then the download process is allowed. This confirms the license acceptance step as a requirement for accessing the project files. An on-chain record of the license terms is created so that there exists a proof or evidence of it.

Data flows both ways from the frontend to the blockchain and the back-end PHP environment and vice versa throughout the entire workflow. The frontend is in charge of real-time interaction, the blockchain secures license and project data immutably, and the backend provides the legal logic necessary for license compatibility decisions. Due to this integration, real-time license compliance enforcement, transparency, and verifiable record-keeping are enabled in the system, while usability and responsiveness for the user are ensured.

4.4 Challenges and Solutions

The system was difficult to develop and implement due to the challenge of integrating blockchain and web development functionality, licensing verification, and reliable function level code tracking.

The primary challenge faced was the manual management of wallet addresses of users. Because blockchains require unique wallet identities for each user interaction, the system right now asks administrators to manually set each user's wallet address. Using the same method in a prototype phase is functional, but is not scalable. Further, it can cause unintentional human error or security risks. A future version with wallet generation automated and decentralized identity management tools integrated would be required for better usability and scalability.

Another big challenge was to ensure various functions' code are accurately extracted and hashed. Detecting and hashing individual functions in different programming languages was complicated and thus reliable regular expressions and a content parser were implemented in JavaScript. Slightly changed formatting or syntax can change the hash outputs which can result in false negatives or positives in compliance checking. To avoid this, we enforced our hashing on the body of a function and ignored other elements like whitespace or comments.

Finally, using the platform on the local XAMPP server, there were some limitations concerning the scalability and deployment of the platform. Local hosting is appropriate for development and

testing work but limits multi-user access. To accommodate future growth, our deployment might need to switch to a cloud platform or decentralized storage system.

Even with the challenges faced, the system successfully shows how blockchain, smart contracts, and function-level code analysis can be incorporated to automate Open Source Software licenses compliance. The implemented solutions will create a strong base for future developments, including automated wallet management, enhanced form-level code analysis, and integration with proprietary software and licensing models.

Chapter 5 - Testing and Results

5.1 Testing Methodology

The system was tested primarily through manual testing via the web interface and browser-based developer tools. The testing focused on validating the system's ability to perform function-level code extraction and hashing, detect license incompatibilities, and interact with the Ethereum blockchain.

Testing was executed by uploading sample C, Java and Python language files, which is currently the only supported file types due to the limitations of the function extraction logic implemented in the JavaScript frontend. The system scans these files, extracts function bodies, generates cryptographic hashes, and checks against previously stored blockchain records.

Commented [GK19]: You should do some testing also with the other languages and also with more complicated files and mention that here too.

Browser debugging tools were used to monitor network calls, inspect blockchain transactions, and verify the correctness of hash generation, license checks, and compliance enforcement. Although no formal automated unit testing was conducted, repeated manual tests were used to ensure end-to-end functionality.

5.2 Test Scenarios and Outcomes

The testing covered a set of realistic scenarios designed to evaluate whether the system could correctly identify reused code, enforce license restrictions, and accurately record transactions on the blockchain. Given the current implementation, all tests were limited to C, Java, and Python source code.

Commented [GK20]: Change this also.

Test Case	Expected Result	Actual Result	Status
Uploading a new C, Java or Python project with no reused functions	Accepted	Project accepted	Pass

Test Case	Expected Result	Actual Result	Status
Uploading a C, Java or Python project reusing GPL functions under MIT	Rejected due to license incompatibility	Upload blocked and user notified	Pass
Uploading a C, Java or Python project reusing functions with compatible licenses	Accepted	Project accepted	Pass
Function hash matching accuracy on C, Java or Python files	Reused function triggers license compatibility check	Matching detected and check executed	Pass
License agreement recorded on download	License stored on blockchain	Confirmed license agreement stored immutably	Pass

Commented [GK21]: That is a nice table but put a caption and maybe mention that similar tests were run for the other programming languages.

Overall, the system functioned as expected when handling all 3 types of files, correctly extracting functions, generating consistent hashes, and enforcing license compatibility checks based on blockchain records and backend rules.

5.3 Test Results and Analysis

The test has shown that the system can automatically check for Open Source Software license compliance within the limits of supporting C, Java and Python source files. The function hashing and license verification workflows worked as expected. As a result, if a prior download and reuse of code resulted in a license incompatibility, the project could not be uploaded.

Using a hashing method that works at the function level, exact matches of the code were found. However, testing also validated the limitation of hash-based verification that we already knew of: a slight change in the logic or formatting of the function results in the creation of a new hash. Consequently, the system treats these altered functions as an entirely new one. The fact that the

hash-based systems face this limitation is known but this happens specifically in the case of Open Source Software reuse and it is a limitation that can be improved upon in the future.

One test result was that the system could not handle any programming other than C, Java and Python. The logic for extracting method and their corresponding regular expressions provided in the tool are made for these languages only, which limits the usefulness of the tool mostly in real-world software ecosystems. To enable support for more languages, we first need to incorporate more sophisticated code parsing or language-specific analyzers.

Commented [GK22]: Do not forget to change also these parts of the text.

The concept of using blockchain for license and project tracking makes sense, as it provides immutability, transparency and auditability. On-chain recording was done for all project uploads and license agreement. Testing has, however, uncovered a drawback: Ethereum gas costs, which are already well-known among blockchain projects. The gas fees did not stop it from functioning during testing, although if we were to scale it up for common use and frequent use, it would not be practical.

In conclusion, the testing validated the system's core concept and technical feasibility within the controlled scope of these language projects. It effectively demonstrates that blockchain-backed license compliance, coupled with function-level code tracking, can automate Open Source Software license enforcement. However, limitations such as language support, reliance on exact hash matches, and gas costs highlight the need for future development and optimization to broaden applicability and robustness.

5.4 Questionnaire and User Evaluation

The questionnaire aims at obtaining user perspectives regarding two key aspects of the project. First, the general perceptions about open source software, licensing, and blockchain technology. Second, expert feedback on the proposed blockchain-based OSS license compliance system. The questionnaire was designed with a conditional structure, adjusting according to the background of the respondent.

Every participant regardless of their technical expertise first answered several general questions that touched on their familiarity and experience with OSS, their opinion on licensing and compliance, and the extent to which they think blockchain and smart contracts can be used to

enhance transparency and rule enforcement. They revealed the general public's understanding and opinion of OSS license management and new technologies.

At the end of this general section, participants were asked whether they had a software development background. If they answered "no," the questionnaire concluded. However, if they answered "yes," they were shown a short demonstration video of the OSS compliance tool developed for this thesis. After watching the demo, these technically experienced participants were presented with a second set of questions aimed at evaluating the tool's interface, usability, clarity, and practical potential. They were also invited to share thoughts on the tool's limitations, supported languages, and possible contexts for deployment. This branching design ensured that general users could provide valuable high-level insights, while developers with relevant experience could assess the implementation itself in depth. Most importantly, the questionnaire aimed to determine whether the use of blockchain and smart contracts is perceived as a viable and trustworthy method for addressing long-standing OSS license compatibility challenges.

The questionnaire contained various types of questions that were structured. It has been generated using Google forms and it was delivered digitally. The design ensures clarity, accessibility, and adaptability to the level of the participant.

The section common to all participants consisted of concept explanation and specific questions targeting the understanding of open-source software, software license, challenges of compliance, and technical aspects related to blockchain technology such as smart contracts. They were made in non-technical terms so all respondents could understand them. Some questions including locating OSS tools that they are familiar with could pick multiple answers and some measured opinion on a standard five point Likert scale.

A conditional branching logic was used; if a participant reported that they have a development software background, they were shown a short demo video of our system. After the video, participants answered more Likert-scale questions, which assessed the usability, clarity, and usefulness of the tool. There were also free-response questions that asked how it could be improved, what features they wanted, and what programming languages they wanted it to support.

The complete questionnaire consisted of 22 questions but varied however many questions each participant answered depended on their background. Encouraging honest feedback responses were collected anonymously.

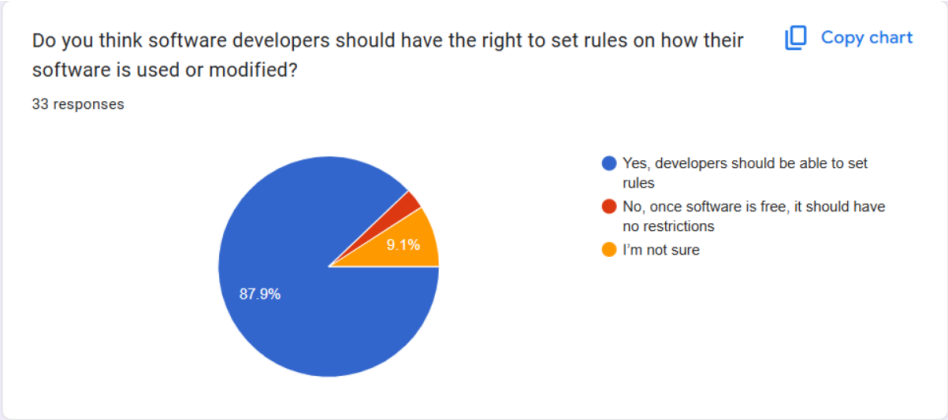
A total of 33 individuals responded to the questionnaire. The sample was intentionally diverse, including participants from both technical and non-technical backgrounds in order to capture a broad spectrum of perspectives on software licensing and blockchain-based compliance tools. Of the 33 respondents, approximately 72.7% indicated that they have a software development background, while the remaining 27.3% did not. This branching distinction determined how much of the questionnaire each participant completed. Non-developers provided feedback on general perceptions of open-source software, licensing, and blockchain, whereas those with a development background were additionally exposed to the system demonstration video and asked to evaluate the tool's implementation and usefulness. In terms of technical roles, the participant pool included.

- 24.2% Computer Science bachelor students.
- 30.3% bachelor students from non computer science disciplines.
- 18.2% junior software developers.
- 18.2% researchers or academics.

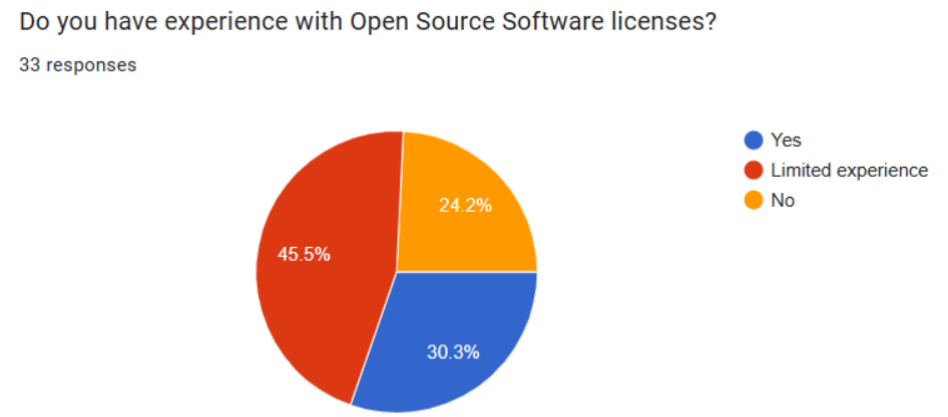
Smaller portions identified as senior developers, software engineers, or professionals in data analysis. This mixture of backgrounds enabled the collection of both high-level user impressions and detailed technical feedback from individuals with relevant domain expertise.

5.5 Questionnaire Results and Analysis

The results of the survey confirmed great support for the principles of this project. Most participants agreed on the importance of developer control, the challenges associated with OSS license complexity, and the lack of effective enforcement mechanisms.

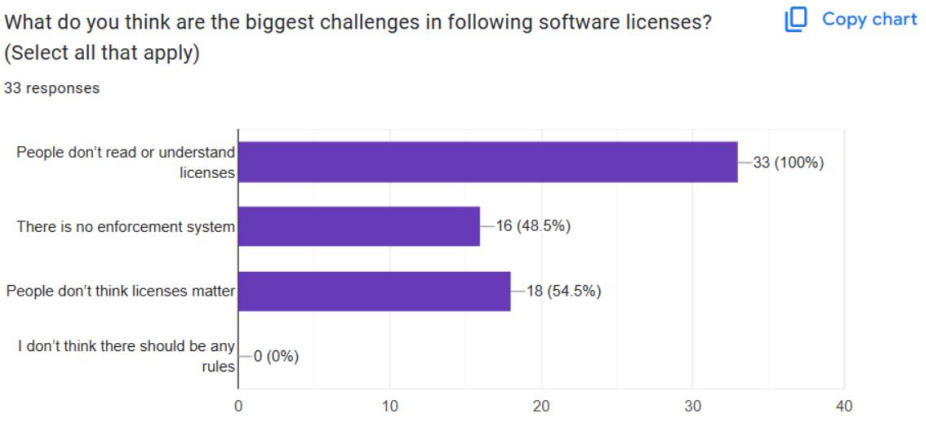


An astonishing 87.9% of respondents agreed that developers should decide the rules regarding how their software is used or modified. It can be said that it essentially relates to the main objective of the system, which is the grant of permit terms by authors and their enforcement using smart contract logic.

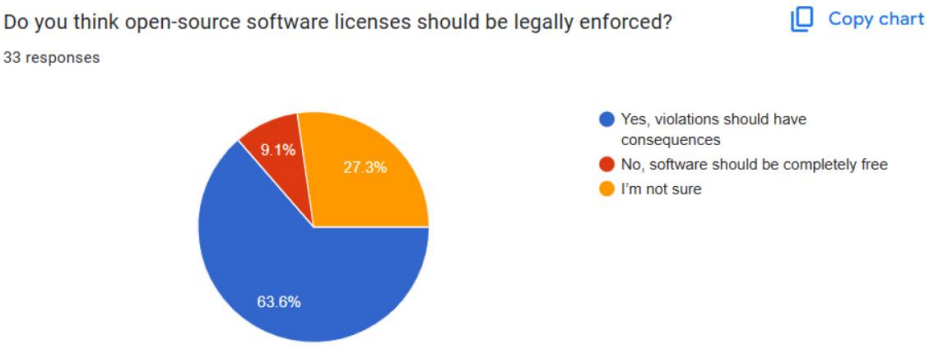


However, when asked about their experience with OSS licenses, only 30.3% reported having direct experience, while the majority had either limited knowledge (45.5%) or none (24.2%). This demonstrates a widespread gap in understanding that can lead to accidental misuse of code — a

gap the system aims to close by automating license verification and flagging violations based on function-level code reuse.



The strongest confirmation of the project's relevance relate came from the response to the question about the biggest challenges in software license compliance. All participants (100%) mentioned that 'people don't read or understand licenses'. Almost half (48.5%) provided absence of enforcement mechanism as a reason. 54.5% replied it is because some people don't take licenses seriously. This shows us the validity of our choice of a blockchain-based platform that offers transparency and immutability.



When asked whether OSS licenses should be legally enforced, 63.6% agreed that violations should have consequences. This reinforces the importance of proactive measures, such as license validation at the upload stage, to minimize legal risk and promote responsible code reuse.

In conclusion, the data back the main argument of this thesis regarding a transparent, decentralized, and automated system that can increase OSS license compliance. By combining blockchain technology with smart contracts, it can address technical enforcement gaps as well as developer community expectations and concerns.

Chapter 6 - Conclusion and Future Work

6.1 Summary of Findings

In this thesis we have presented a blockchain-based solution to Open Source Software licensing compliance through smart contracts, function-level code analysis and automatic license compliance checks. The users can upload and download a software project and the code is reused for always respecting the license obligations.

By using function-level hashing, the platform more precisely flags code reuse than what is currently offered by traditional license compliance tools. The Ethereum blockchain securely and immutably records all license agreements and the project's metadata. This approach helps create a clear trail of compliance.

According to testing, the system effectively prevents any user from uploading software under incompatible licenses when previously downloaded code is used. The license checks were successfully executed by the function hashing mechanism for C, Java and Python files. The system's conceptual design was validated as license agreements and project uploads were recorded onto the blockchain.

6.2 Contributions of the Thesis

This research contributes a novel approach to Open Source Software license compliance by:

- Demonstrating the feasibility of function-level tracking of code reuse,
- Automating license compatibility checks before redistribution,
- Integrating blockchain smart contracts to enforce compliance and preserve licensing records immutably.

By focusing on proactive enforcement rather than post-distribution audits, the system shifts the compliance process upstream, reducing legal risk and developer uncertainty.

6.3 Future Work.

Although the system serves its fundamental objectives, testing and analysis revealed some areas of improvement for the future. Due to the nature of the design of the function extraction logic, the

current implementation is limited to C, Java and Python language files. A possible elaboration could be adding support for additional programming languages like JavaScript, C++, C# and many more. This will increase the use case of the system and will be more relevant. Future experiments could be performed on integrating abstract syntax tree (AST) analysis or machine learning-based code similarity detection to increase robustness with respect to minor function alterations. The manual handling of wallets is a limitation to scalability. Future versions must have automated wallets and decentralized identities for improved usability and reduced management costs. Moreover, although gas costs of Ethereum did not obstruct testing, they do pose a barrier to scaling the system for many users. Finally, while the architecture of this prototype is specifically geared towards open-source software, it could be expanded to proprietary software compliance. Integrating payment systems and licensing fees could enhance the accessibility and commercial viability of the platform. Its use in a large organization also needs to be tested, in order to examine the scalability of the approach, also across organizations.

This thesis shows how code and blockchain technology combined can help automate or enforce compliance of Open Source Software licenses using fine-grained code analysis. While there are challenges left as there is a certain coverage of programming languages and cost of the blockchain system, the model offers a good solution for greater legal certainty and transparency in software reuse.

Commented [GK23]: change this also

References

[1] Brock, A. (Ed.). (2022). Open source law, policy and practice (2nd ed.). Oxford University Press.

[2] Lindberg, V. (2008). Intellectual property and open source: A practical guide to protecting code. O'Reilly Media.

[3] Rosen, L. (2004). Open source licensing: Software freedom and intellectual property law. Prentice Hall.

[4] Sarmah, S. S. (2018). Understanding blockchain technology. Independently published.

[5] St. Laurent, A. M. (2004). Understanding open source and free software licensing. O'Reilly Media.

[6] Kapur, R., Briggs, M., Saha, T., Costa, U., Carvalho, P., Chong, R. F., & Kohlmann, P. (2010). Getting started with open source development. IBM Corporation.

[7] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[8] Szabo, N. (1997). Formalizing and securing relationships on public networks. First Monday

[9] Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An overview of blockchain technology: Architecture, consensus, and future trends.

[10] Solidity. (n.d.). *Solidity Documentation*. https://docs.soliditylang.org

[11] Nomic Foundation. (n.d.). *Hardhat: Ethereum development environment*. https://hardhat.org

[12] Ethereum Foundation. (n.d.). *Ethereum Official Documentation* https://ethereum.org/en/developers/docs/

[13] Mozilla Developer Network. (n.d.). *JavaScript Guide* https://developer.mozilla.org/en-US/docs/Web/JavaScript

[14] PHP Group. (n.d.). *PHP Manual*. https://www.php.net/manual/en/

[15] Oracle Corporation. (n.d.). *MySQL 8.0 Reference Manual*. https://dev.mysql.com/doc/refman/8.0/en/

[16] Ethereum JavaScript API. (n.d.). *Web3.js Documentation*. https://web3js.readthedocs.io/

[17] Apache Friends. (n.d.). *XAMPP for Windows*. https://www.apachefriends.org/index.html

[18] Zohar, A. (2015). Bitcoin: Under the hood. Communications of the ACM.

[19] Georgia M. Kapitsaki, Nikolaos D. Tselikas, Ioannis E. Foukarakis: An insight into license tools for open source software systems. Journal of Systems and Software 102: 72-87 (2015)