Academic Year
2024-2025

Computer-aided conflict detection between open-source software licenses

Victor van Herel

Master's thesis
Master of Science in computer science: computer networks

Supervisor
prof. Prof. Serge Demeyer, AnSyMo, UAntwerpen
Supervising assistant
ing. M. Beyazit, AnSyMo, UAntwerpen

University of Antwerp
Faculty of Science

Contents

List of Figures

List of Tables

Chapter 0

Preamble

Abstract

Open source projects use software licensing as a tool to control how their software, and resulting forms such as compiled binaries, are used. However, people interacting with open source projects often do not have a background in law, and these software license texts are inherently legal in nature. This mismatch becomes relevant when project maintainers attempt to introduce code under a foreign license into their project, as the joined working of said license if often not immediately as easy to understand.

This thesis proposes a method to help alleviate this problem in a large number of cases, and draw attention to more cases which may cause problems. It does this by employing large language models to extract characteristics of licenses which can be easily understood by project maintainers, thus removing the need for a costly legal consultation.

Acknowledgments

To the community of Space Station 14, without whom this thesis would not have taken shape.

Chapter 1

Introduction

This thesis focuses on open source software licenses, we clarify this by defining this as the practice of placing restrictions on the use of a project's code and derived forms such as compiled binaries. Oftentimes, this is done through the use of standard licenses which are made to be shared and reused. But sometimes a project maintainer may choose to make their own license entirely. This is possible due to the way authorship rights work, which is the first subject we will shortly examine.

1.1 Authorship rights & open source

Whenever someone creates a work, which can take any shape and includes software projects, and fixes it in a shared form, they become the author and owner of all rights over the project. This means that, without the need to register the work with any organization or institute, the author can control all aspects of how their work may be used. Importantly, this also implies that no other person that obtains the work is allowed to use it without an explicit permission grant [1, 2, 3].

This is a stark contrast to open source projects. For this, we start by examining the Open Source Definition, a set of ten requirements for being Open Source, as defined by the organization with the same name [4]. Without going over each requirement in particular, the goal of publishing a work in an Open Source way can be described as allowing anyone to benefit from the work provided by allowing general "use" of the work.

In the software world, this takes a very concrete form, namely sharing of code online which others are allowed to use in their own projects subject to the restrictions imposed by the author.

1.2 Standardized licenses

This is where standardised licenses come in, of which there are many [5, 6, 7, 8]. Standard licenses intend to offer a sensible set of defaults for project maintainers to choose from which have this Open Source idea built in. In fact, there is a number of licenses which the Open Source organization explicitly confirms as compliant to their definition [6].

Most of these licenses are maintained by license stewards, and it is important to recognize that these stewards are usually organizations which have the ability to perform legal analysis of their licenses [6].

Additionally, their widespread use allows them to be analyzed and argued about which further expands the legal basis and understanding of these documents.

A well known license which is very easy to understand is the MIT license, included as an example [9]:

```
Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the ‘‘Software’’), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ‘‘AS IS’’, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

This license has a clear structure and is very easy to interpret. This thesis will call back to this license (and others) in Chapter 3, where the interpretation will be elaborated on further. To anyone reading it however, the intention is clear. Anyone in possession of work under this license is allowed to perform any action with it, so long as they maintain the license and copyright statement, as well as the liability disclaimer in capitals.

1.3 Software integration patterns & the definition of related work

The choice of a license is a decision usually made quite easily, and in most cases the project maintainer succeeds in their goal: Allowing anyone to use their work in ways they define by their choice of license. However, the project maintainer may in turn decide to want to make use of another project themselves in their project.

If the piece of code the project maintainer wishes to use is licensed differently, problems can occur. Sometimes these issues become high-profile, especially when large organizations are involved, which signifies the relevance of the mismatch between legal knowledge and software maintaining knowledge [10, 11, 12].

At this point it is important to take a step back and understand how licenses deal with the action of working forwards on the work of someone else. In summary, this has the following legal grounds:

- **U.S. copyright law:** "A “derivative work” is a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work of authorship, is a “derivative work”." [1]

  This definition is explicit, and defines the term "derivative work" to signify work that is based on or otherwise derives from existing work from another author. It is important to note the explicit callout that derivative work only applies if the work itself is an original work of authorship.

- **EU directives:** EU law does not have a specific definition to use, but does use the concept of adaptation and transformation in their law, which applies to the practice of using shared code [2].
- **International treaties:** A document which is relevant internationally in countries that undersigned it is the Berne Convention for the Protection of Literary and Artistic Works. This convention asserts protection for works that are adapted as follows:

  "Translations, adaptations, arrangements of music and other alterations of a literary or artistic work shall be protected as original works without prejudice to the copyright in the original work." [3]

This summary of the legal perspective on deriving from existing works makes clear that there are many different perspectives on how to handle these situations.

1.3.1 Application on software

When we examine the specific domain of software however, we can identify two patterns of software integration which hold relevance to software licensing. For brevity, we refer to "the source" as the repository in which code to be derived from is found. We refer to "the target" as the repository in which the derivation will be used.

- **Full inclusion:** Whenever a piece of code, even mere lines of code, is taken verbatim from a source and placed into a target, the license of the source continues to hold over that piece of work, but separation between where the application of each license happens may not be very clear.

  Licensing conflicts arise here in the form of grants that the source's license requires, which are not guaranteed by the target license. For example, the GPL's copyleft requirement which requires that deriving work remains licensed under a similar license is therefore not compatible with non-matching licenses in this way.

  Actions that produce full inclusion are among the following:

  – Cherry-picking: Either through `git␣cherry-pick` or other similar command that transfers a commit or other section of work verbatim into the target repository, this produces a scenario in which work is included from another source.
  – Forking: Splitting off from a work as a fork of that work takes the entire work and then allows you to make your work from it. You can choose to license your changes differently, so long as the original license permits this.
  – Binary inclusion: Even binary forms of software are protected under the license which holds over the source code, unless that license specifically disclaims it. In this case inclusion of the binary form of the source's code in the target works the same as the above mentioned cases.
  – **Generally:** Any direct inclusion of source material in any form in any form of the target, even when modified, qualifies as a full inclusion, **so long as separation between the coverage of each license in play is unclear**. It should be noted that it does not matter where the work comes from, whether that be a repository with a clearly posted license, or a website like StackOverflow or other online forum boards and similar. Even if the license there is not immediately clear, you will need permission from the author to use that work in the way you intend to use it.

  This aligns with the definition of "derivative works" in U.S. Copyright Law [1, 13, 14, 15].

- **Separated interaction:** Code can rely on other pieces of code in a separated way. In this sense, we primarily mean the interaction pattern of a library as an example. Here it is important to note that the works are fully separate in source code, unless the source code is fully included in the target repository, which would make it a full inclusion.

  This thesis does not examine this pattern in detail, as it is generally less restrictive than the full inclusion pattern. If a license configuration is compatible in the full inclusion pattern, it will also be compatible in this configuration. If it isn't, it may become compatible in this configuration.

  In practice, this pattern arises when any of the following is done:

  - Dynamic linking of binary forms: The binary forms are separate and can also be delivered separately. For all intents and purposes, they are separate aside from the fact that one cannot function for its intended purpose without the other.
  - High-level language dependencies: High level languages such as Java, Python, JavaScript, ... have dependency managers (Maven, Pip, NPM, ...). These allow a developer to define dependencies on libraries declaratively in a file. Automated tooling can read these files and configure the project's dependencies on one's device fully independently. Sometimes this pertains source code, sometimes this pertains binary code. However, for both cases, the source's work is kept fully separate from the target's work and only interacts with it via high-level system calls.
  - **Generally:** Any target work that requires a form of source work to function, but which doesn't need to come bundled with it in any form, as the source work can be obtained separately.

  The United States Copyright Act definition which matches the most to this pattern is a "collective work". It is defines as: "a work [...] in which a number of contributions, constituting separate and independent works in themselves are assembled into a collective whole" [1, 15].

It should be noted that in highly collaborative projects, it is quite easy to come into contact with the full inclusion pattern. The separated interaction pattern is ubiquitous, as any form of library usage qualifies. This means that whenever this happens in any way, shape or form, the project maintainer performing said action must be aware of which licenses they interact with. We return to the initial statement that project maintainers are not necessarily well-versed in legal matters and reading legally constructed licenses, which results in a problem.

1.4 Research statement

When we consider the concept of licensing however, we must understand that we are still dealing with natural language. This means we can attempt to apply natural language processing techniques to these licenses to learn properties and induce automated reasoning about licenses, and providing supported advice to project maintainers when handling licensing issues.

This thesis is a feasibility study into one of those techniques using recent advances in large language models (LLMs for short). Specifically, we aim to answer the following questions:

- **Research question 1:** Which properties of software licenses are relevant to critically reasoning about combination compatibility?

  While license texts are composed of various legal elements, it is important to examine whether or not answering the compatibility question depends on all of these elements. If we can define a subset of important properties for correct identification of most software licensing conflicts, we can focus efforts on automatically determining these properties instead of performing complex per-combination reasoning.

If these properties exist, this will also assist in allowing actors outside of the legal profession to understand why a specific conflict occurs in a general and understandable sense, as only the property needs to be explained, not the in depth text of the license itself.

- **Research question 2:** Is it feasible to deploy large language models to determine these properties of a license?

  If the ground work is laid by answering Research question 1, the next step is to automate it. This thesis examines the option of using large language models to do so, and intends to compare these results against known values. This is the ground truth which we have decided to build on, which will be explained further in Chapter 3.

  If automation of this process can be made reliable, existing automated workflows can be amended with this method to further improve license compliance reporting.

Chapter 2

Practical context

Before we attempt to fomulate an answer to the questions proposed in Chapter 1, we first need to consider the context we are working in in a more concrete sense. This chapter examines existing standards and tools which are used by the industry to interact with software licenses, conflicts between them, compliance, ... right now.

2.1 Accepted standards

To start, we would like to provide an overview of accepted standards in the industry which provide a leading effort in helping to handle complex software licensing interactions. These standards are listed in no particular order.

- The **SPDX (System Package Data Exchange)** standard describes itself as "An open standard capable of representing systems with software components in as SBOMs (Software Bill of Materials) and other AI, data and security references supporting a range of risk management use cases.". To summarize, it is a way of including detailed licensing information right along with source code of the project in a standardized and well-defined way. This empowers projects such as ScanCode and FOSSology, which we will examine in the next section, to make very accurate reports of projects that employ the standard. It can also be employed for other use cases which are not relevant to this thesis, such as software security documentation.

  Aside from this, as a necessary addition, the SPDX standard also defines a set of licenses along with their license text which is immutable per published version of the standard. This is necessary as one of the standard's primary goals is facilitating software license compliance in a precise way. Having a fixed set of licenses to work with which each are provided with a fixed ID allows us to refer to licenses in very specific and unambiguous ways, provided they are part of the standard [16, 5, 17].

- The **ReUse Software** project is an initiative by the Free Software Foundation Europe which provides a similar solution, profiling itself as an easy way to perform software license management in open source projects specifically. Like the SPDX standard, it provides tooling to add licensing information on a very granular level (up to individual files) within an open source code project. It improves the amount of projects automated scanning software can reliably analyse, with a focus on usability by project maintainers.

  This project does not provide a separate list of licenses, as it opts to make use of the SPDX license list [18].

These standards are primarily practical ways with which individual project maintainers can improve legal legibility of their software code, and offer this thesis a stable source of license texts in the form of the SPDX license list, along with a way to refer to each license correctly in short.

As an example, the MIT license simply has the identifier `MIT`.

2.2 License scanners

To perform license detection on projects that do not use either standard (though the listed scanners do fully support reading those), scanner software exists that generates reports on what licenses are in use in the project either directly or indirectly. The way these work differs, but the goal is the same, expanding the license horizon beyond what an individual project lists about itself.

- The **ScanCode** project focuses on providing machine-readable compliance reports as well as human-readable reports on a software component's licensing information which includes that of its dependencies. It is able to analyse source code and binary files for these license and copyright statements and is actively used in the industry to visualize compliance information and improve discovery of problems related to it. It should be noted that it generally does not provide automated reasoning about license configurations outside of policies defined by its user [19].
- Similarly the **FOSSology** project also offers the same scanning functionality, but focuses on workflow-based deployments rather than being able to be used ad-hoc and directly by project developers. It is targeted at actors within the industry for whom license compliance is a primary concern [20].
- **Licensee** is a light-weight tool which does not provide a suite for gaining insight in how licenses are used in the project, including dependencies. Instead, its focus is correctly identifying license texts to an SPDX license ID in pre-determined locations. Usually the `LICENSE` file of a repository [21].
- Other scanners do exist as well, but are usually part of enterprise-level all-in-one solutions. The license compliance part operates in generally the same way.

A key fact to consider is that while these scanners allow project maintainers to gain insight into their license usage, and even in some cases provide tooling to automatically check against certain policies the user defines, they do not provide any automated reasoning on their findings. If an incompatible license combination exists within a scanner's report, it is the responsibility of the user to flag it [19, 20]. It is then a logical conclusion that the work presented in this thesis could be a first step to further improving the capabilities of these scanner projects.

2.3 Custom licenses

While repository maintainers can choose to use a pre-defined license (often part of the SPDX license list), we recall that this action is, in a general sense, the person choosing the terms under which others can use their work. This does not have to be by choosing a pre-defined license, as one can instead decide to create their own.

While this is an option that is not often taken, as popular platforms such as GitHub encourage the use of a license upon creation of a repository to store code in [22], custom licenses do exist and our thesis considers these as well. We show this fact by considering two avenues of reasoning:

- The GitHub platform provides information on the number of repositories it hosts, and other metadata which can be consulted publicly. In this metadata, a project's primary license is listed as well. When one queries this data, an "Other" category is displayed which ranks third globally. This category contains all licenses which could not be identified by Licensee, the scanner which GitHub employs. This indicates that there is a very large number of repositories which use licenses that aren't known to the dataset it uses [7].

- Widely used licenses were once defined by someone. As a result, within lists of publicly available licenses (SPDX, ScanCode, ...), we can often recognize names that refer to companies or industry actors. Licenses like the Pixar license, the radvd license, the PostgreSQL license, the Ruby License, ... are all examples of licenses that were made dedicated to a project, and may have gained traction and use in other projects since. This indicates that developers do indeed choose to make their own license if they cannot find an existing license that suits their need [5, 23].

2.4 License families

While many different licenses exist, we can group them together into different license families based on their characteristics. This allows us to reason about groups of licenses in a general sense. The families of licenses relevant to this thesis are described here.

2.4.1 Copyleft licenses

A license that belongs in this family has a copyleft clause. This is a clause that requires that any derivative work created from the licensed work is also licensed under the same license, thus providing the work and any derivatives to the general public in an irrevocable way.

The point of copyleft licenses is to ensure the openness of the work by use of a license. Copyleft licenses require an author to make the work open source, and due to the reciprocity effect, any derivative works must also be made open source in turn under the same terms. This is a very strong requirement, and a sub-family which weakens this effect is the Copyleft Limited family, which does require source code redistribution for derivative works, but the obligation to redistribute source code of linked projects, some of which may be proprietary, is limited to provisions present in the license itself [23, 13, 14].

Well-known examples of the strong copyleft license family are the GPL and AGPL licenses. The Copyleft Limited license family is represented by the well known LGPL licenses [23].

2.4.2 Permissive licenses

The permissive licensing model creates an alternative approach to openness of the software. It does not impose the same limitation of which license derivative works must fall under. These licenses usually only require that the license text is retained along with the covered code in both source code and binary form, if those are provided to the general public and the covered code is contained within. This is as such a very open family of licenses, which places very reasonable conditions on usage of the work, but once satisfied, allows a wide range of permissions.

The permissive license family varies a lot, and as such there are no clear categories within it. These are not necessary however, as generally, two permissive licenses can work with each other, unless if provisions exist that conflict in both licenses [13, 14]. It should be noted that in the next chapter, we will find that for within the ground truth of this thesis, no combination exists where two permissive licenses are incompatible with eachother.

Chapter 3

State-of-the-art: OSADL

TOFIX ►*Victor: This chapter contains a lot of uncurated work, and the order or relevance of everything presented is not yet fixed in place. Here be dragons.*◄

The Open Source Automation Development Lab (OSADL) is a collaborative project that supports and promotes the use of open-source software, particularly Linux, in industrial and automation environments. OSADL focuses on ensuring that Linux and other open-source software components meet the stringent requirements of industrial applications, including real-time capabilities, legal compliance, and long-term maintenance.

Among the key activities of the OSADL organisation, **license compliance** is a dedicated subject it examines. OSADL helps member companies ensure their use of open-source software complies with licensing obligations by providing tools and services for legal audits and documentation that go deeper than those listed in Chapter 2 [24].

This aspect of the OSADL organisation is very interesting for this thesis, as it provides this dataset to the public for analysis and automated interaction. We will examine each of the relevant components.

3.1 OSADL License Checklists

The OSADL organization describes this project as follows: "Whenever Open Source software is copied and distributed which typically is permitted by every type of Open Source license, a number of obligations and prohibitions are imposed on the distributor. It is very common for the recipients of such software to recursively redistribute it in such a way that a chain of distributors and recipients is created – all of them having to fulfill the same license obligations. However, for the time being, there is no common understanding of how these obligations are to be fulfilled in detail which regularly leads to misunderstandings, conflicts and sometimes even to court cases.

This project is launched with the goal to generally establish checklists of obligations of commonly used Open Source software that are accepted by distributors and copyright holders and trusted by all members of the distribution chain." [8]

This description is executed by creating license checklists at the request of its members for specific licenses. These checklists provide an overview of all the obligations someone who wishes to use work licensed under it must fulfill, in a standardized way. For this, it uses several building blocks which we describe below. It should be noted that all raw data for each component is publicly available, and downloading this can be automated.

While it is possible that a checklist is formatted in a JSON format, and OSADL publishes a schema to validate against for these objects, we use the text format for explaining its workings in this thesis.

It is important to frame the checklists in the scope in which they are provided, quoted directly from the OSADL project page: "The checklists assume a situation where a licensee of

Open Source software incorporates such software components into a product - either a physical device with installed software or a software distribution on a storage medium or on the Internet - and needs to establish appropriate processes in order to fulfill the imposed license obligations for legal compliance when conveying the product to customers." [25]

This maps directly to the Full inclusion integration pattern we discussed in Chapter 1, limiting the use in particular to re-use of code and modification, as well as redistributing the completed result in some way.

3.1.1 Checklist language

Language used for checklists is fixed. It uses only defined terms, which sometimes allow freeform input (for example license names). These language constructs are separated into three main categories:

- **Language elements:** Core elements of language that convey a particular meaning. These elements appear inspired by the language definitions used for RFC documents, as provided in RFC 2119. This inspiration however is not publicly cited on the web page. These elements include, but are not limited to, terms such as YOU MUST, YOU MUST NOT, USE CASE, ... Notably, the OSADL authors have chosen to make these elements all uppercase to allow them to be distinguished easily.
- **Actions:** These elements are usually verbs, and are usually, but not always, what follows a language element in a provision. A couple of examples include terms such as: Publish, Provide, Add, Append, ...
- **Terms:** These elements follow usually follow an action in provisions. This is the largest set of defined elements among these categories.

In effect, these allow us to write obligations in near-English sentences. For example, everyone understands "YOU MUST Provide License text" to mean just that, the requirement to provide the license's text if you use the covered work. How exactly these statements interact with each other is what we will cover next.

3.1.2 Format description

Example: The MIT license

To explain how the format works, we explain this by referring to an example checklist of the MIT license. That is provided verbatim in text form as follows.

```
USE CASE Source code delivery OR Binary delivery
 YOU MUST Provide Copyright notices
 YOU MUST Provide License text
 YOU MUST Provide Warranty disclaimer
```

We refer back to the MIT license text which was included in Chapter 1. Let's pick this checklist expression apart by listing all used definitions first, taking them verbatim from the dataset OSADL provides:

- **USE CASE:** Sometimes the license obligations may allow the distributor to freely select between a number of optional use cases; the USE CASE language construct is introduced for this purpose. Several USE CASE language constructs to which the same license conditions apply may be combined using the OR language construct. If a particular USE CASE is mentioned repeatedly, e.g. once along with another USE CASE and once not, the obligations of all USE CASE sections must be fulfilled.

- **YOU MUST:** The YOU MUST language construct specifies an individual license obligation, i.e. what to do, probably among other things, to become license compliant. It may optionally be followed by indented language constructs such as ATTRIBUTE that further describe the license obligation.
- **OR:** When the OR language construct is used between elements of a condition, then the condition already applies, if only one element is fulfilled. When the OR language construct is used between obligations, then it is sufficient to fulfill at least one obligation. Note: The AND language construct is assumed by default between consecutive license obligations and attributes.
- **Provide:** The action to Provide means to make available particular material such as a license text to another natural or legal person. While the action to Forward is restricted to conveying existing material, the action to Provide expands the meaning in the sense that the material may be newly generated as long as it fulfills its purpose.
- **Source code delivery:** Licenses may treat the various aggregate states of deliverable software such as source, intermediate and object code differently. The term Source code delivery denotes a situation where source code is delivered, and no software component is included in the delivery without corresponding source code.
- **Binary delivery:** A software distribution may contain material not in a human-readable programming language, but in binary machine or intermediate code that was generated from the project's source code using a compiler. Some licenses may then impose disclosure obligations that may be fulfilled either by delivering the corresponding source code along with the binary code or by offering to do so at a later date. Irrespective of whether disclosure obligations exist and how they must be fulfilled, when software is delivered at least partly in object form then this is referred to as Binary delivery.
- **Copyright notice:** The Copyright notice indicates the name of the holder of the exclusive usage rights. In its minimal form, it may only contain a name in an obvious context. Usually, however, the name of the holder of the exclusive usage rights is preceded by the word 'Copyright', the © symbol or the letter c in parentheses '(c)', and a number or several numbers that indicate the year when the work was created." If the author is not the holder of the exclusive usage rights, the name of the holder of the exclusive usage rights may be followed by an attribution to the author.
- **License text:** The term License text denotes the unabbreviated original text of a particular license in its original language. It may either be printed on paper or contained in a data file on a medium using an obvious and well-known or an individually defined and specified character encoding.
- **Warranty disclaimer:** The license text may contain a clause or several clauses where the original authors refuse any warranty for malfunction or damages that may occur when using the licensed software. Such section is referred to as Warranty disclaimer.

With these definitions provided, we can pick apart the checklist line by line.

- `USE␣CASE␣Source␣code␣delivery␣OR␣Binary␣delivery`: The subordinate elements of this checklist apply when distributing source code or binary forms of a project which is, in full or in part, subject to the terms of the license this checklist was made for.
- `YOU␣MUST␣Provide␣Copyright␣notices`: In these use cases, you are required to provide the copyright notice.
- `YOU␣MUST␣Provide␣License␣text`: You must also provide the unaltered license text.

- `YOU␣MUST␣Provide␣Warranty␣disclaimer`: And finally, you must provide the warranty disclaimer.

Thus, this checklist tells us exactly what we need to do when we want to redistribute work licensed under the MIT license in any code-bound form. Complying with all three requirements is as simple as including the full text of the original license, which includes the copyright notices (first line of the license) and the warranty disclaimer (last paragraph of the license in capitals).

This explanation aligns with our explanation in Chapter 1 about how the MIT license works, and demonstrates how the format works.

Formal definition

The OSADL project also defines this format formally, where it is split in two sections. The first section of the format is displayed in the example, and contains these checklist items. We call this the primary section.

TOFIX ►*Victor: Fix this section. What is the goal?*◄

3.1.3 Detecting license family based on a checklist

OSADL also provides insight into how to determine the license family of a license. We recall from Chapter 2 that this thesis considers each license to be part of one of two families: Permissive, or copyleft to any degree.

TOFIX ►*Victor: Fix this section as well.*◄

```
USE CASE Source code delivery
 YOU MUST Provide License text
 YOU MUST NOT Modify Copyright notices
 IF Software modification
  YOU MUST Grant License
   ATTRIBUTE Original license
  YOU MUST Provide Copyright notices
 IF Commercial distribution
  YOU MUST Indemnify Other contributors
USE CASE Binary delivery
 YOU MUST NOT Modify Copyright notices
 EITHER
  YOU MUST Provide Source code
 OR
  YOU MUST Provide Delayed source code delivery
   ATTRIBUTE Inform Recipient
  EITHER
   ATTRIBUTE Customary medium
  OR
   ATTRIBUTE Via Internet
   ATTRIBUTE Reasonable
 IF License change
  YOU MUST Use Identical License obligations
  YOU MUST Use Warranty disclaimer On behalf of Other contributors
   ATTRIBUTE Effective
  YOU MUST Use Liability disclaimer On behalf of Other contributors
   ATTRIBUTE Effective
  IF Service offerings
```

```
    ATTRIBUTE NOT Transferable
  IF Commercial distribution
   YOU MUST Indemnify Other contributors
```

3.1.4 Copyleft table

OSADL also provides a JSON table that indicates for each license whether or not it has a copyleft clause. This value can be one of the following:

- **Yes:** This indicates the license is part of the Copyleft Strict license family.
- **Yes (restricted):** This also logically implies the license is part of the Copyleft Limited license sub-family. Together with the licenses tagged Yes, this forms the full Copyleft license family.
- **No:** This maps to the permissive license family.
- **Questionable:** This final result implies that OSADL cannot come to a consensus on the presence of a copyleft clause in the license, thus implying that no general consensus exists. At the time of writing, this is assigned to the MS-PL and OpenSSL licenses in the dataset.

3.1.5 Compatibility matrix

OSADL also provides a curated matrix of compatibility between licenses, based on their checklists. This matrix is an aggregate product of the data elements and is therefore not new. The way these decisions are made however is not clearly documented.

Before we look into the specifics of this matrix, as it allows us to draw some valuable conclusions already, it is important to recall the scope in which the license checklists, which this matrix was based on, are constructed. This context was given earlier in this chapter.

Important definitions that result from combining two of these licenses in the scope of a compatibility question are the following:

- **Leading license:** This is the license of the project that is the target of the inclusion. Code from the subordinate will end up in the project under this license.
- **Subordinate license:** This is the license that covers the code that is being taken.

To conclude, a compatibility result in the compatibility matrix provided by OSADL references a situation where one incorporates a part of the subordinate project into a leading project, and where the licenses of both projects are individual. This is still a case of the full inclusion pattern.

A compatibility result is either Yes, No, Unknown or Check Dependency. The first three are clear, but the Check dependency result needs to be clarified. OSADL introduces the concept of depending compatibility which defines that two licences are indeed incompatible, but allow a change to be made to the license configuration to make them compatible. This is to be read explicitly as: The configuration itself is **incompatible**, but one or both of the licenses allow themselves to be changed out for another license **explicitly** which does result in a compatible scenario.

Expressing percentages

In total, the dataset contains 116 licenses, which results in 13456 total combinations. However, we are only interested in 13340 of these combinations, as 116 of them are licenses being compared with themselves. These combinations are always compatible, and OSADL reports them as "Same." for this reason.

Looking further, we see that 1020 of the combinations are listed as "Unknown". This leaves us with 12320 assessed combinations of which:

- 8812 ($\sim 71.53\%$) are compatible.
- 3448 ($\sim 27.99\%$) are not compatible.
- 60 ($\sim 0.48\%$) fall in the case of depending compatibility.

An interesting finding arises when we combine this with the copyleft table, mapping "Yes" and "Yes (restricted)" to Copyleft, and "No" to Permissive. These results are shown below:

Table 3.1: OSADL compatibility in a copyleft-oriented view.

Subordinate \Leading	**Copyleft**	**Permissive**
Copyleft	812 total, 75 yes, 60 check dependency, 677 no	All 2465 no
Permissive	1729 total, 1498 yes, 231 no	All 7140 yes

This table shows a powerful property: If the leading license is permissive, we only need to know the license family of the subordinate license to be able to assess compatibility. **We conclude already that the license family is an important license property to automatically determine compatibility as part of the answer to RQ 1.**

Additionally, we see that a Copyleft/Copyleft combination is primarily incompatible, with some edge cases. These are always cases where the license lists licenses it is explicitly compatible with. For example, GPL-1.0-or-later is compatible with all future versions of the license by definition. (But not the other way around! Compatibility is a one-way relationship.)

Lastly, we observe that Copyleft/Permissive combinations are primarily compatible, but a not insignificant portion of combinations are incompatible.

To conclude this section, we can now make the powerful claim that if we know the license family a two licenses belong to, we can already answer $\sim 78.67\%$ of combinations, i.e. those where the leading license is permissive.

3.2 Inducing the Permissive Leading effect logically

In this last section, we present a logical argument as to why a permissive leading license allows for direct classification of combinations based on the license family the subordinate license belongs to. To do this, we consider the following definitions by OSADL: TOFIX ►*Victor: These need to be moved earlier. Also this needs to be argued about why these are disjoint.*◄

1. **Compatibility is assumed, if:**
   - compatibility with the other license is explicitly ruled in a particular license, or
   - the two licenses in question both do not contain a copyleft clause, or
   - the leading license contains a copyleft clause and the other license does not and also does nto impose any obligation that the first license does not allow to impose.
2. **Incompatibility is assumed, if:**
   - incompatibility with another license is explicitly ruled in a particular license, or
   - one license imposes an obligation that the other license does not allow to impose, or
   - the two licenses in question both contain a copyleft clause and no license contains an explicit compatibility clause for this license combination.

These definitions already directly indicate why Permissive/Permissive is always compatible, namely it is defined as option 2 of OSADL's definition of compatibility.

This only leaves us to clarify why Permissive/Copyleft results in incompatible combinations at all times. To do this, we need to argue that these combinations always meet at least one of the three criteria for incompatibility.

To do this, we first recall what it means for a license to be in the copyleft license family: This means derivative works based on the covered work must be licensed equally in the strict case, or similarly in the limited case.

- In the strict case, we land in bullet point two of incompatibility immediately. As we are trying to license the derivative work differently from the subordinate license.
- In the limited case, the same bullet point is reached as generally, the conditions under which the copyleft requirement may be waived do not include the full inclusion pattern. Alternatively, if the license is Copyleft Limited because it allows a specific subset of licenses to be used, we observe that these other licenses are also all in the Copyleft family, thus reverting to the base case in this bullet point.

As such, it is only logical that attempting to incorporate work licensed under a copyleft license using full inclusion in a work that is licensed under a permissive license creates an incompatible construction.

Chapter 4

Method / Experimental Design

4.1 LLM selection

TOFIX ►*Victor: Describe what OLLama is. Describe why each model was chosen.*◄

4.2 Important property: Copyleft clause presence

We call back to Chapter 3, where we determined the presence of a copyleft clause, and thus the license family to which a license belongs, as an incredibly powerful indicator for determining further license configuration compatibilities. To put this into practice, a proof of concept is proposed for an experiment to determine a license's family using the license text, a pre-composed query and one of the selected models. We will evaluate this using accuracy, with our ground truth values being taken from the OSADL copyleft table.

4.3 Approach

With the models selected, we simply query the model for each license to obtain its answer to the question: Does this license text contain a copyleft clause?

This allows us to extrapolate from these results and assign the license to a license family. We also have the ability to check this evaluation against a ground truth provided by the OSADL, which allows us to assess an accuracy score for each model.

In order to address stochastic behavior which is inherent to large language models, we will query each model multiple times. This thesis has chosen to do this 5 times for each model, to weigh practicality of inference time with results.

The model's decision for a given license can then be retrieved with the majority vote of each run.

The query each model is presented with for each run is the following, where the license's text is inserted in the indicated position:

```
=== LICENSE FULL TEXT ===

{license_fulltext}

=== INSTRUCTION ===

You are a license compatibility expert. Does the license contain a copyleft clause?
A copyleft clause is a provision that requires derivative works to be distributed
```

```
under the same license terms as the original work, ensuring that the freedoms
granted by the license are preserved in derivative works.
Begin your answer with a yes or a no for easy parsing.
```

The license full text used is fetched from the SPDX license database, because of its comprehensive nature and ability to have fetching be fully automated based on SPDX license identifiers [5]. The SPDX license database is the most comprehensive authority of license texts which are meant for public use. Other tools such as ScanCode and FOSSology use SPDX license identifiers to refer to matched licenses [19][20].

Lastly we remark that this query poses a yes/no question which limites the LLM to generating binary responses only. This differs from the OSADL dataset described in Chapter 3, which also includes the option "Questionable". We provide this limiting to the LLM to ensure it provides an answer every time, rather than defaulting to the neutral answer.

Chapter 5

Method results

5.1 Copyleft clause detection results

We consider the results of the copyleft detection experiment first as follows:

Llama3:8n: Upon a first run of this model with the given task, it was decided not to continue with the additional runs for this model, as its accuracy was already very low and as such Llama3:8b is not a promising avenue compared to other selected models. The confusion matrix for this run is provided in Table 5.1. This run scores an accuracy of **55.17% (64 / 116)**. While the mistakes made by this model are varied, it is clear the model particularly struggles with false positive detections of a copyleft clause.

Table 5.1: Llama3:8b confusion matrix (Run 1/1)

LLM \OSADL	Copyleft	Permissive
Copyleft	25	48
Permissive	4	39

Gemma3:4b: This model performed better, scoring an average accuracy across runs of 83.62% (97/116). Its mistakes are also exclusively failures to detect a present copyleft license. The confusion matrix for the majority vote is given in Table 5.2, which scored the same accuracy as the average accuracy. The model is very consistent, only differing in responses across runs for the MPL-1.1 license and the MPL-2.0-no-copyleft-exception license.
Average accuracy: 83.62% (97 / 116)
Majority-of-5 accuracy: 83.62% (97 / 116)

Table 5.2: Gemma3:4b confusion matrix (Majority vote of 5 runs)

LLM \OSADL	Copyleft	Permissive
Copyleft	10	0
Permissive	19	87

Deepseek R1:8b: This model is very inconsistent with its answers, answering differently across runs for 28 licenses in the dataset. Additionally, this model classifies 2 specific licenses incorrectly on all its runs. We will discuss this rather peculiar result after the listing of direct results. In a majority configuration however, we note that the accuracy increases (107 licenses correctly classified, as opposed to 104 in the best run). This suggests that this model does benefit from a majority vote setup. Notably, this model also does not make any false positive detections.
Average accuracy: 88.62% ($\tilde{1}$02.8 / 116)
Majority-of-5 accuracy: 92.24% (107 / 116)

Table 5.3: Deepseek R1:8b confusion matrix (Majority vote of 5 runs)

LLM \OSADL	Copyleft	Permissive
Copyleft	20	0
Permissive	9	87

- Qwen3:8b: This model is the best model of the selected models for this task. It however doesn't really benefit from the majority vote decision setup, like we have seen for the Deepseek model. Instead, its runs generally are very accurate, outclassing all other 3 models that were included in this experiment, however the best run actually performs better than the majority vote, correctly classifying 113 out of 116 licenses. We see in the confusion matrix that this model does make false positive classifications, as opposed to the other two models which do not make this mistake.

Average accuracy: 96.21% ($\tilde{1}$11.6 / 116)
Majority-of-5 accuracy: 96.55% (112 / 116)

Table 5.4: Qwen3:8b confusion matrix (Majority vote of 5 runs)

LLM \OSADL	Copyleft	Permissive
Copyleft	26	1
Permissive	3	86

5.2 Examining mistakes made by models

5.2.1 General mistakes

The Gemma3, Deepseek R1 and Qwen3 model all failed to classify two licenses correctly, looking into the thinking of the models when making this decision, we can explain why:

- **IPL-1.0:** The IBM Public License is a complex case because it handles source code redistribution and binary object redistribution entirely differently. We examine specifically its Section 3 Requirements. This mentions the following:
  - "When the Program is made available in source code form: a. it must be made available under this Agreement; and b. a copy of this Agreement must be included with each copy of the Program." This section tells us that source code redistribution is strongly copyleft, requiring the same license is used for the derivative work created.
  - "A contributor may choose to distribute the Program in object code form under its own license agreement, provided that: [...]" This section governs binary delivery, and it does impose that whichever license agreement you use, it complies with the IPL-1.0, and must disclaim warranty on behalf of the contributors, excluding them from liabilities, damages, ... much like the MIT license. It goes on to permit the new license to differ, but that such different clauses are only offered by the redistributing contributor alone. This difference between handling of source code redistribution and binary object redistribution seems to confuse our models. The option to redistribute under a different license, given specific requirements, is not a copyleft clause. As such, the models decide to classify this license as permissive in all runs.
- **Sleepycat:** The sleepycat license is also always classified incorrectly. In this case, the source of confusion is very obvious. The sleepycat license itself is actually 3 licenses which were concatenated together. Because of this, and the predictable nature of the format of these licenses, the models get confused when running inference, treating each license referenced as a separate question. As a result, it fails to comply with the answer format provided, meaning automatic detection of the answer fails as well for Deepseek R1 and

Qwen3. For Gemma3, it actually is capable of considering the entire license as its own block, however it is mistaken in handling of the license itself, missing the fact that the Sleepycat section of this license is indeed BSD-3 Clause, but adds an additional clause which imposes a Copyleft clause to this format.

5.2.2 Individual mistakes

An interesting mistake only made by the Qwen3 model occurs when examining the Artistic-2.0 license. Gemma3 and Deepseek R1 remain faultless here. This mistake seems to stem from Clause 4a in the license, which *is* a copyleft license, but is preceeded by a construct that indicates this is optional if compliance is instead ensured with clause 4b or clause 4c in the license, which are not copyleft clauses. In a sense, Qwen3 correctly answers the question posed, as this simply queries the model for a Yes/No answer to the question: "Does this license contain a copyleft clause?". This differs from being a part of the copyleft license family.

Other mistakes do not appear to form a coherent pattern. Additionally, among the list of most used licenses according to GitHub's Innovation Graph project, mistakes still get made by Gemma3 and Deepseek R1 within this subset. Qwen3 remains faultless within this subset across all of the recorded runs [7].

This list of frequently used licenses is included in this document as follows in order of rank provided: MIT, Apache-2.0, GPL-3.0-only, GPL-3.0-or-later, AGPL-3.0-only, AGPL-3.0-or-later, BSD-3-Clause, GPL-2.0-only, CC0-1.0, Unlicense, MPL-2.0, BSD-2-Clause, CC-BY-4.0, LGPL-3.0-only, LGPL-3.0-or-later, CC-BY-SA-4.0, ISC, MIT-0, BSD-3-Clause-Clear, EPL-2.0, WTFPL, EUPL-1.2, 0BSD, BSL-1.0, Zlib, EPL-1.0, MulanPSL-2.0, UPL-1.0, OFL-1.1, Artistic-2.0

It should be noted that the dataset of OSADL-evaluated licenses does not contain all of the above mentioned licenses. In particular, it does not contain: CC0-1.0, CC-BY-4.0, CC-BY-SA-4.0, BSD-3-Clause-Clear, MulanPSL-2.0, OFL-1.1

Impact of this finding: We conclude by considering the impact of this finding. In particular, this finding is interesting as we can already eliminate a lot of possible combinations with this result. We can do this because OSADL's definition of Compatibility between licenses contains a specification that two licenses which do not contain a copyleft clause are compatible. Likewise, OSADL's definition of Incompatibility stipulates that two licenses containing a copyleft clause without presence of an explicit compatibility clause in the license text.

As an experiment, this allows us for the list of popular licenses above to immediately classify 156 possible license combinations as compatible correctly. It also allows us, ignoring explicit compatibility clauses, to classify 94 of the possible license combinations as incompatible. The actual number is 70 however, as this algorithm is not aware of explicit compatibility clauses. This calculation assumes a total of 576 combinations examined, of which 24 are combinations of the same license, in which compatibility is also assumed.

Chapter 6

Conclusion

The first paragraph of the conclusion is usually a short review of this paper/thesis goals, problems, and context.

After that we generally cover the following main points in the next paragraphs:

- **Summary of Results.** In a paper/thesis, we probably have many pages in previous sections presenting results. Now in the conclusion, it is time to put the most important results here for the reader. Especially research with measurable results, we highlight the numbers here.
- **Main Findings / Conclusions.** Many times, we have a result but based on its number we can draw a conclusion or formulate a finding on top of it. Even if it was previous discussed in an earlier section, we need to re-state here.
- **Contributions.** If we presented/discussed the main contributions of this research in the introduction, then we need to do again in the conclusion. Do not repeat verbatim what was written in previous sections. In the conclusion, we expect contributions to be more detailed and linked to the results/findings when possible.

Avoid generic conclusion sentences that could be applied to anything. For example, "Our technique showed good results which were beneficial to answer our research questions. Our work can be used by other researchers to better understand our domain." Instead, go for more specific detailed results. For example, "Our technique showed a precision of 75% which was 15% higher than the baseline comparison. Based on this we can see that ..."

The final paragraph (or paragraphs) of the conclusion is about future research. We can create a separate subsection for it if there are multiple paragraphs dedicated to future work. Just be aware, it is not a good sign if future research content is longer than what we wrote for the previous paragraphs in the conclusion.

Bibliography

[1] 17 u.s. code § 101 - definitions. https://www.law.cornell.edu/uscode/text/17/101. Accessed: 2025-06-24.

[2] Directive 2001/29/ec of the european parliament and of the council of 22 may 2001 on the harmonisation of certain aspects of copyright and related rights in the information society. https://eur-lex.europa.eu/eli/dir/2001/29/oj/eng. Accessed: 2025-06-24.

[3] Berne convention for the protection of literary and artistic works. https://www.wipo.int/treaties/en/ip/berne/. Accessed: 2025-06-24.

[4] The open source definition. https://opensource.org/osd. Accessed: 2025-06-22.

[5] Spdx license list. https://spdx.org/licenses/. Accessed: 2025-06-22.

[6] Licenses approved by the open source initiative. https://opensource.org/licenses. Accessed: 2025-05-21.

[7] GitHub. Github innovation graph. https://github.com/github/innovationgraph, April 2025. Dataset. Released under CC0-1.0.

[8] Osadl open source license obligation checklists homepage. https://www.osadl.org/OSADL-Open-Source-License-Checklists.oss-compliance-lists.0.html. Accessed: 2025-04-13.

[9] Mit license text. https://opensource.org/license/mit. Accessed: 2024-11-22.

[10] Linux kernel mailing list message on the subject of blocking tuxedo computers from interacting with gplv2-only modules. https://lkml.org/lkml/2024/11/14/709. Accessed: 2024-11-27.

[11] Licensing issue on the tuxedocomputers/tuxedo-drivers gitlab repository. https://gitlab.com/tuxedocomputers/development/packages/tuxedo-drivers/-/issues/137. Accessed: 2024-11-27.

[12] Licensing issue on the archived tuxedocomputers/tuxedo-keyboard github repository. https://github.com/tuxedocomputers/tuxedo-keyboard/issues/61. Accessed: 2024-11-27.

[13] Halina Kaminski and Mark Perry. Open source software licensing patterns. https://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=1009&context=csdpub. Accessed: 2024-10-21.

[14] Ravi Sen, Chandrasekar Subramaniam, and Matthew L. Nelson and. Determinants of the choice of open source software license. *Journal of Management Information Systems*, 25(3):207–240, 2008.

[15] Daniel M. German and Ahmed E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *2009 IEEE 31st International Conference on Software Engineering*, pages 188–198, 2009.

[16] The system package data exchange (spdx). `https://spdx.dev/`. Accessed: 2025-08-20.

[17] Phil Odence. Why you should use spdx for security. `https://www.linux.com/news/why-you-should-use-spdx-for-security/`, January 2023. Accessed: 2025-08-20.

[18] Reuse software. `https://reuse.software/`. Accessed: 2025-08-20.

[19] Scancode documentation. `https://scancode-toolkit.readthedocs.io/en/stable/getting-started/home.html`. Accessed: 2024-12-18.

[20] Fossology homepage. `https://www.fossology.org/features`. Accessed: 2024-12-18.

[21] Licensee, a ruby gem to detect under what license a project is distributed.

[22] Kedasha Kerr. Beginner's guide to github repositories: How to create your first repo. `https://github.blog/developer-skills/github/beginners-guide-to-github-repositories-how-to-create-your-first-repo/`, June 2024.

[23] Scancode licensedb. `https://scancode-licensedb.aboutcode.org/`.

[24] Open source automation development lab (osadl) eg. `https://www.osadl.org/`. Accessed: 2025-04-13.

[25] Checklists scope. `https://www.osadl.org/Checklists-scope.oss-checklist-edit-scope.0.html`. Accessed: 2025-04-13.

Appendix A

Relation with Research Projects

As part of my master courses I participated in a series of research projects. Here I list how far these overlap with my master's thesis.

- **Not applicable.**