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:** To what extent is deploying large language models to determine these properties of a license feasible? By extension, to what can this setup accurately assess a given license combination's compatibility?

  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.

  The question aims to provide a general set of recommendations regarding the feasibility of one such system in particular, based on metrics determined against these known truths.

  If 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

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 these key activities of the OSADL, **open source license compliance** is a key area it provides support in to its members. It does this in a number of ways that are of great interest for this thesis, with the goal of helping member companies and organizations in complying with obligations imposed by open source licensing and the way it interacts with their work. These tools it develops go a lot deeper than what the projects listed in Chapter 2 aim to accomplish in certain areas [24].

3.1 Open source license checklists

A first element we examine is the open source license checklists. At the time of writing, the OSADL provides what can be interpreted as a checklist of obligations to comply with when one wishes to interact with a specific license, for a set list of 116 licenses. This checklist has a relatively fixed format, which we will go over shortly before moving on to the next topic.

An open source license checklist is essentially formulated as a checklist with items to ensure software license compliance in general cases, using well-defined language elements. A checklist has two sections, an obligations section which describes the obligations imposed by a license and an optional supplements section which contains supplemental information which is sometimes already encoded in the obligations section.

It is important to understand that these checklists are still the result of manual curated work, as they are produced on the basis of requests made by members of the OSADL organization. Especially the supplements section, which often lists things like the presence of a copyleft clause, explicit (in)compatibilities, ... is the result of expert knowledge weighing in on the formulation.

The OSADL also makes clear in which context these license checklists are used. On this subject, it says the following: "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]

To draw a parallel to the integration types discussed in Chapter 1, this lines up fully with the full inclusion pattern, due to the sentence "... incorporates such software components into a product ...".

3.1.1 Example: The MIT license

To explain how the format works, we explain this by referring to an example checklist of the MIT license, shown below:

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

The MIT license is an example of a license that does not need a supplements section, as it is very simple in scope and application, and does not list any specific interactions with other specific licenses. Still, we can learn some key facts from this about a license checklist's structure.

- All obligations listed are "scoped" in one or more use cases, preceeded with the `USE CASE` statement. We will discuss the different use cases in a separate section.
- Obligations can be very specific in what they impose. In Chapter 1, we showed the text form of the license. Here we see that this license is split up into the copyright notice (the first line), the license text (lowercase paragraph) and the license disclaimer (uppercase paragraph). Each element is handled separately (though identically) in the license checklist.

It should also be noted that the OSADL does define each of the terms it uses (with some exceptions in the case of license names, ...). For example, the following definitions exist quoted directly from the OSADL data:

- **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.
- **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.

In these definitions, we see the three separated elements from the MIT license returning as defined constructs which are defined generally, and thus also applicable to other license models.

3.1.2 Detecting copyleft clauses based on a license checklist

An interesting property of these license checklists is that, while always explicitly mentioned in the supplements section, a copyleft property can also be inferred through the obligations section. We show this property with an example, namely the GPL-1.0-only license.

```
USE CASE Source code delivery
    YOU MUST Provide Copyright notices
        ATTRIBUTE Highlighted
        ATTRIBUTE Appropriately
    YOU MUST Provide Warranty disclaimer (Warranty disclaimer)
        ATTRIBUTE Highlighted
        ATTRIBUTE Appropriately
    YOU MUST NOT Modify License notices
    YOU MUST NOT Modify Warranty disclaimer (Warranty disclaimer)
    YOU MUST Provide License text
    IF Software modification
        YOU MUST Grant License
            ATTRIBUTE Original license
        YOU MUST Provide Modification notice
        YOU MUST Provide Modification date
        IF Interactive
            YOU MUST Display License announcement
            YOU MUST Display Copyright notices
            YOU MUST Display Warranty disclaimer
            YOU MUST Reference License text
    YOU MUST NOT Restrict Granted rights
```

(Continued on next page.)

```
USE CASE Binary delivery
    YOU MUST Provide Copyright notices
        ATTRIBUTE Highlighted
        ATTRIBUTE Appropriately
    YOU MUST Provide Warranty disclaimer (Warranty disclaimer)
        ATTRIBUTE Highlighted
        ATTRIBUTE Appropriately
    YOU MUST NOT Modify License notices
    YOU MUST NOT Modify Warranty disclaimer (Warranty disclaimer)
    YOU MUST Provide License text
    EITHER
        YOU MUST Provide Source code
            ATTRIBUTE Machine-readable
    OR
        YOU MUST Provide Written offer (Written offer)
            ATTRIBUTE Duration 3 years
            ATTRIBUTE To Any third party
            ATTRIBUTE No profit
            ATTRIBUTE Delayed source code delivery
                ATTRIBUTE Machine-readable
    IF Software modification
        YOU MUST Grant License
            ATTRIBUTE Original license
        YOU MUST Provide Modification notice
        YOU MUST Provide Modification date
        IF Interactive
            YOU MUST Display License announcement
            YOU MUST Display Copyright notices
            YOU MUST Display Warranty disclaimer
            YOU MUST Reference License text
    YOU MUST NOT Restrict Granted rights
(... 73 license compatibility/incompatibility statements ...)
COPYLEFT CLAUSE Yes
```

(Explanation of copyleft clause detection on next page.)

In this example, the supplement (represented by the last two lines) mentions correctly that a copyleft clause is indeed present. However, we see this requirement returning in the obligations section, namely in the following lines:

```
USE CASE Source code delivery
    ...
    IF Software modification
        YOU MUST Grant License
            ATTRIBUTE Original license
    ...
USE CASE Binary delivery
    EITHER
        YOU MUST Provide Source code
            ATTRIBUTE Machine-readable
    OR
        YOU MUST Provide Written offer (Written offer)
            ATTRIBUTE Duration 3 years
            ATTRIBUTE To Any third party
            ATTRIBUTE No profit
            ATTRIBUTE Delayed source code delivery
                ATTRIBUTE Machine-readable
    IF Software modification
        YOU MUST Grant License
            ATTRIBUTE Original license
        ...
```

When summarized like this, we see a strict copyleft requirement emerging. Namely, in both delivery use cases, you are required to grant your changes under the original license (**but not the original copyright notices**). Even more so, in Binary delivery, you are also obligated to provide source code, either immediately or in the form of a written offer valid for 3 years. This last obligation forces you into the Source code delivery use case in both cases, where the copyleft requirement for the source code also holds.

Logically, to comply with the obligations in this checklist which represent the license itself, regardless of what you do, you must provide your derivative work in source code form and optionally binary form under the same license. (But again, not the original copyright notices.)

3.2 Notable use cases

Before we continue with other ground truth data sources the OSADL provides, we take a moment to consider the other noteworthy use cases the OSADL defines and uses in their license checklists. For clarity, these are the "arguments" to a USE CASE statement in the checklist.

First, we recall that we already identified what "Source code delivery" and "Binary delivery" means. With this in mind, the following terms of interest remain:

- **X delivery Of Combined library:** The OSADL defines a Combined library as follows: "If two or more libraries are copied into a single library that offers the functionality of the library components all together, but without applying any modifications to them, then the resulting library is referred to as a Combined library."

  As a result, delivery (both binary and source code) of such a work fall under this work, and are used in the LGPL-family of licenses. This family is a non-strict copyleft license, and as such mentions libraries explicitly.

- **X delivery of Combined work:** Likewise, the OSADL provides the following definition for the term Combined work: "A Combined work is created when a work consists of two or more originally separated components that are combined in such a way that they form a new work no longer allowing them to operate independently."

  This has a similar implication, but is more general on one hand, not limiting itself to libraries, but also more specific in the sense that the work must not be able to be separated as a result. Regrettably, which functions are used as a benchmark to determine this treshold of functionality is left rather vague. We see this use case also used by the LGPL family of licenses.

- **Combined work delivery:** The Sleepycat license is the only license to use this use case, and it does so to enforce the copyleft property on what the OSADL defines as a Combined work (see the bullet point above). In this use case, one is required to either include the source code in the combined work, or to provide delayed source code delivery of combined work for no profit, similarly to the GPL-1.0-only example previously handled. Note however that this license does not define a time window in which this delayed delivery must be performed. We will return to the Sleepycat license later, as it appears to be a difficult license to work with in particular.

- **Binary delivery of Linked work:** The OSADL defines a Linked work as: "A work may be linked to another work at compile time or at run time in which case the other work is referred to as Linked work."

  Examples of this arise in any programming language when one uses a library in their code without including the library itself as a baked-in option. (For example, `requirements.txt` for Python, module imports for Go, classpath linking for Java using Maven or Gradle, or any other pattern in which the dependencies of a project are described in a declarative way, based on which automated tooling can independently retrieve the library files needed itself.) The **library** is the linked work.

  This has a variant, namely **Binary delivery of Linked work With Header files OF Library Included In Linked work**, which exists to suit the more specific requirements of particular licenses it is used in.

  Once again, the LGPL family uses these terms explicitly. It is also used by the Artistic-2.0 license.

- **Font, Image, Work, ... delivery:** The delivery word can be used with a number of various other terms, indicating that the license in question covers this "type" of work separately.

  These terms are used by more general licenses that don't specifically cover code, like the CC family of licenses. However, we also see these elements appearing in the Bitstream-Vera and the OFL-1.1 license.

  As the term "Work" is very general, we call out its OSADL definition explicitly to clarify it: "Copyright law specifies a Work as any artifact that was created by a human being, can be perceived by a human being and possesses individual characteristics that are attributable to a human being. In the context of software licensing, the term Work is used to denote a collection of software components that are conveyed as a whole under the same license or, if compatible, under the same group of applicable licenses."

- **Network service:** This does not constitute delivery in the traditional sense, rather, the OSADL defines it as follows: "The term Network service is used to describe a situation when a provider does not convey a software application in binary form to a customer, but installs it on a network-capable computer and let customers use the application by communicating with it via a network link."

This use case is outlined explicitly by the AGPL family of licenses, which is known for specifically differing from the GPL family of licenses because it covers the "network service loophole". This use case is also explicitly covered by the AFL-3.0 and the OFL-3.0. It should be noted that this use case is generally understood not to be covered by any variation of the term "Derivative work" in copyright law, as one is using the work themselves and in doing so, is providing a different service to the end users.

3.3 Copyleft table

Aside from the now examined license compliance checklists, the OSADL makes available a distilled source of information. Namely, a table that lists the copyleft status of each of the licenses in the raw data. It should be noted that while the licenses themselves list the copyleft property in their license checklist supplements, this table is actually more specific.

This is because this table goes deeper than a yes/no answer. Namely, it lists the following possibilities:

- **No:** Licenses with this classification are permissive.
- **Yes:** Licenses with this classification are Strict Copyleft family.
- **Yes (restricted):** Licenses with this classification are in the Copyleft Limited family.
- **Questionable:** This classification is used for two licenses, namely the MS-PL and OpenSSL licenses. As these licenses are not entirely clear in their copyleft state, and their interactions with other licenses are generally not defined, this thesis will not use these licenses and any information derived from them as ground truth.

3.4 Compatibility matrix

Based on these license checklists, we can consider a situation in which two license checklists are checked against eachother, one in a leading position and the other in a subordinate context. This context is derived from the context of an individual license checklist. Recall, that a license checklist itself is made in the context of the licensed work being included in another work, without specifying what the other work is licensed under.

In this section, we examine the situation that occurs when we do know what the including work is licensed as. In this sense, we define the leading license as the license that governs the including work. The work that is being included, in part or in full, is governed under the subordinate license.

This construction is a construction that the OSADL has already examined for a lot of the licenses it covers. It provides the raw data of these experiments, which are still the result of human curation, as a matrix which we can examine. In this matrix, we have the following types of results:

- **Yes:** This result indicates that the obligations imposed by both licenses do not conflict. However, the combined work still must abide by both licenses' obligations. As such, this compatibility is only theoretical, not practical.
- **No:** This result indicates incompatibility. This means there is some obligation combination that causes problems when applied.

  As we will see, the copyleft property of a license is an important thing to know here. For instance, if the leading license is permissive, and the subordinate license is copyleft in any way, then by definition the combination cannot be compatible. This is because the

combination would become a derivative work of the copyleft licensed work, thus requiring it as a whole to be similarly licensed.

- **Check dependency:** A third option is what the OSADL defines as Depending compatibility. This is a combination of two licenses in the context previously discussed, which in the definition of the two participating licenses is incompatible. However, either one of the licenses allows it to be replaced at any time with another license it specifies, usually a newer version of the same license, which does make the combination compatible. These cases are rare, and are flagged separately. As an example, an LGPL license can generally be upgraded into its GPL variant, dropping the special provisions surrounding linking. LGPL licenses are not compatible with GPL licenses, but naturally GPL licenses are compatible with themselves.
- **Unknown:** Not all license combinations were evaluated by the OSADL. In this case, the field is listed as this value.

3.4.1 OSADL definition of (in)compatibility

Given these possible classifications, it is useful to tie this into the OSADL's definitions of compatibility and incompatibility. Shown below are both definitions, copied verbatim:

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

It should be noted that the OSADL presents these clauses as an unordered list. For our purposes however, we numbered these lists such we can refer to C1, C2 and C3 as the compatibility cases, and I1, I2 and I3 as the incompatibility cases. We introduce this naming, such that we can answer the following question:
An interesting point to examine is whether or not:

1. All license combinations are covered by these definitions. (Completeness property)
2. A given license combination belongs in at most one definition, either Compatibility or Incompatibility. (Mutual exclusion property)

To prove the first question we present a logical induction. Firstly, we use the presence of a copyleft clause. On this, it can be stated that a license either has one, or not. This is a property that holds true for any given license. With this property, we have enough information to classify each category verbosely, presenting a combination as LeadingLicenseCategory-CopyleftLicenseCategory:

- Permissive-Permissive: Compatible by C2.

- Permissive-Copyleft: Incompatible by I2, namely, when code licensed under a copyleft license is reused, it must be similarly licensed. This is never true in a project that uses a permissive license.
- Copyleft-Permissive:
  - Incompatible obligations imposed by subordinate license (permissive): Incompatible by I2.
  - No such obligations imposed: Compatible by C3.
- Copyleft-Copyleft:
  - Explicit compatibility ruled: Compatible by C1.
  - No such compatibility ruled: Incompatible by I3.

Lastly, if either license explicitly rules itself compatible or incompatible with another, this is covered by clauses C1 and I1.

Conlusion: The definitions of compatibility the OSADL uses are complete. Every license combination can be classified with these licenses into at least 1 class of compatibility.

Now we analyze the second question, and provide a theoretical example as to why both definitions can apply at the same time. Consider the following licenses:

Theoretical leading license:

```
Copyright (c) COPYRIGHT_HOLDER

Permission is granted to use this licensed work in any form, source code or derived,
for any purpose, so long as it does not involve any of the following:
- Advertising in the name of the copyright holder or contributors to the work.
- Commercial use of the software or other for-profit ways of handling it.
```

Theoretical subordinate license:

```
Copyright (c) COPYRIGHT_HOLDER

Permission is granted to use this licensed work for commercial for-profit purposes
only. It may be used in any form.
```

While these licenses are theoretical, and especially the second license has no real-world representation, these licenses are still both permissive licenses, as neither contains a copyleft clause. This means they match both clauses C2 and I2, as the second license's commercial nature contradicts the first license's non-commercial use limitation.

As such, this theoretical combination of licenses is both compatible and incompatible. However, since stating incompatibility is generally a stronger statement, this combination is as a whole incompatible. Additionally, this theoretical example is not represented by any combination in the matrix.

Conclusion: The OSADL definitions of (in)compatibility are not mutually exclusive, however, incompatibility logically supercedes compatibility in cases where both apply. No combinations where this is the case exist in practice, but a theoretical combination can be made to satisfy both C2 and I2 easily.

3.4.2 Identifying compatibility classes of interest

Given the ground truth information in this compatibility matrix and the use of the definitions in the section above, it is useful to examine which information we need to correctly classify most of license combinations within it. This will in turn allow us to perform automated querying for licenses in a smart way, without having to resort to the infeasible per-combination query behavior. By means of a spreadsheet which we add specific columns to, we can attempt to identify useful classes of interest.

If we make this spreadsheet with one row for each combination, and add two columns indicating the OSADL-provided copyleft license table entry for the leading and the subordinate license, we can already make some important distinctions:

1. **Permissive-X** (9775 / 13340)**:** When the leading license is permissive (= No in the copyleft table), it is immediately obvious that we can produce an accurate decision if we know the copyleft state of the subordinate license. Namely, if it is permissive, the combination is always compatible. If it is any form of copyleft, it is always incompatible. If the subordinate license is classed as Questionable, the result is unfortunately always unknown. Aside from this, this class does not have any Unknown classifications.

   In numbers, we see this class containing the following results:

Table 3.1: Table for Permissive-X compatibility class

Permissive-Permissive		**Permissive-CopyleftAny**		**Permissive-Questionable**
Yes	Unknown	No	Unknown	Unknown
7140	0	2465	0	170
9775 total combinations (out of 13340 ∼ 73.276%)				

2. **CopyleftAny-Permissive** (2465 / 13340)**:** When the leading license is copyleft, and the subordinate license is permissive, generally, we expect the combination to be compatible unless there is some specific cause for it not to be. This is because permissive licenses are generally permissive in how the work covered under them are used, and thus any conflicts should arise from the copyleft leading license disallowing something that the permissive license still requires in terms of material reproduction. This is supported by the number of records in each category, provided as follows:

Table 3.2: Table for Copyleft-Permissive compatibility class

CopyleftAny-Permissive		
Yes	No	Unknown
1498 (∼ 60.993%)	231 (∼ 9.493%)	736 (∼ 29.858%)
2465 total combinations (out of 13340 ∼ 18.2478%)		

3. **CopyleftAny-CopyleftAny** (812 / 13340)**:** This compatibility class is intuitively the hardest to predict. This is because the copyleft property has correctly caused many of the conflicts in the matrix, but we must also recall the fact that certain licenses are defined to be compatible with another license regardless in their license text. This is true in for example versioned licenses, where an old license can be upgraded to a newer version of the same license. This versioned license pattern is exclusive to licenses the OSADL deems copyleft.

   Recall also the introduction of the depending compatibility definition, which we expect to see as well in this case.

4. **CopyleftAny-Questionable** (58 / 13340)**:** This compatibility class is not very significant for this thesis, as it is the product of uncertainties. However, it is still necessary to be

Table 3.3: Table for Copyleft-Permissive compatibility class

CopyleftAny-CopyleftAny		
Yes	No	Check dep.
75 (∼ 9.236%)	677 (∼ 83.374%)	60 (∼ 7.389%)
812 total combinations (out of 13340 ∼ 6.0870%)		

mentioned as the fourth compatibility class to make the union of all compatibility classes a complete overview of all possible combinations on the matrix. Naturally, this can be expanded to licenses not on the matrix as well, though here we do not have a ground truth reference about the copyleft property for these unreferenced licenses.

Within this category, we find mostly Unknown entries. This is a natural consequence of the subordinate license having a Questionable classification. The explanation field of these entries specifically identifies that this answer is given because a different, more exact, answer can only be given in specific cases which go beyond the scope of the general recommendations the matrix is meant to provide.

Interestingly there is 17 entries of the 29 with the OpenSSL license as the subordinate license in this category that do have a definite No answer. However, their explanation is the same explanation given to all other entries in this category where OpenSSL is the subordinate license. Namely the following: "Interpretation: The final statement of the appended SSLeay License 'The licence and distribution terms for any publically available version or derivative of this code cannot be changed' normally must be interpreted as a copyleft clause, but there is no general consensus on this interpretation. Therefore, a recommendation on the compatibility of the OpenSSL license can only be given for selected cases."

As such, we believe this to be a mistake from the OSADL. Either a more detailed explanation exists why these 17 cases do have a definitive answer and it is not given, or they are mistakenly classified as "No", where no detailed explanation as to why exists. This bundled together with the fact that all other license combinations that land in this category are also classified as Unknown makes this a category of uncertainty. Due to this, and the fact that this category only represents 0.4% of all license combinations in the matrix, this is where we conclude our discussion of it for this thesis.

Chapter 4

Method

For this thesis, we will be interfacing with large language models through the Ollama tool. We've selected this tool as we've found it to be very easy to use, providing a simple way to download models and subsequently querying them manually, or through an API which we can use in our reproduction package to automate certain tasks in obtaining our results [26].

For our methods proposed, we will perform this method on multiple LLM's to gauge their aptitude, and potentially also identify characteristics that an LLM should have to be able to give mostly correct answers. Where possible, we will aim for high decision accuracy on the question: Is a given combination of two licenses, where one is the leading license and the other the subordinate license, generally compatible? We do this to be able to dynamically construct a matrix in the same format of the license compatibility matrix provided by the OSADL.

4.0.1 LLM selection

For this thesis, we examine the LLM's outlined below. A general selection criterion was their availability. If an LLM is not openly available to run offline (with necessary preceeding downloads allowed), it is not the subject of this thesis as the goal is to provide a method that is open, and which anyone coming across the project could run on their own.

Aside from this criterion, we also clarify specific reasons for choosing a specific model as follows:

- **Llama3:** A model which is indicated as being lightweight and general-purpose. This model does not feature a thinking property, which will generally make it faster than other models that do feature this property [27]. We include these non-thinking models in the proof-of-concept of this thesis to provide an indication of the trade-off between accuracy of prediction and prediction speed.

- **Gemma3:** This model is more recent compared to the Llama3 model, and as such it is a reasonable expectation for this model to perform better. Like Llama3, this model is a non-thinking model which is advertised to be capable of general-purpose language processing tasks [28].

- **Deepseek R1:** Deepseek R1 is the first model with a thinking property on this list. We also note that it is one of the first open models that feature this thinking capability [29]. Generally, we expect this to be one of the better performing models because of its supposedly expanded reasoning capacity.

- **Qwen3:** The Qwen3 model is the most recently published model in the list of models this thesis uses. Like Deepseek R1, it features a thinking propert. It also advertises enhanced logical reasoning capacities, which are useful given the logical nature of the problems this thesis poses and seeks a solution to [30].

4.0.2 Handling LLM stochasticity

By design, large language models provide stochastic responses. This means that not every response generated by a model given an identical input will produce the same output, which provides a challenge to handle these responses automatically. How can we be certain that a model's answer is correct given the fact that it may answer differently given enough times to query the model identically?

This thesis will used a method inspired by ensemble classifiers in clustering problems to lessen the effect a stochasticly uncertain response may have on further reasoning based on it.

To do this, we will run each query-model combination multiple times, asking it questions which have a closed set of potential answers as much as possible. Where this isn't possible, such as with producing a list of license names, manual curation is needed to produce this closed answer set. For the example, this would be listing the SPDX license ID's for the license names produced. The final answer that will be regarded as the definitive answer of the model is the majority vote of the model's different responses over the answer sets.

A variable to consider is the amount of times a model is presented an identical query for this majority vote system. We will discuss the way this variable is determined in the next section. Importantly, we define the answer of a model in this setup as the **Majority-of-X** answer, where X is the number of queries used in determining said answer.

4.1 Experiment 1: Copyleft clause

A first question this thesis aims to provide a proof of concept for is the ability for large language models to discern whether a given license text contains a copyleft clause. We do this as we recall that, in Chapter 3, we determined that the presence of a copyleft clause, and thus the license family to which a license belongs, is a strong indicator for determining further license configuration compatibilities.

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 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 across the entire dataset. It should be noted specifically, that we will **not** be considering the licenses of which the OSADL provides the answer: Questionable. We will still allow the models to assess these licenses, but for the purposes of calculating accuracy, we don't include these results as there is no ground-truth answer to compare to.

In order to address stochastic behavior which is inherent to large language models, we will query each model multiple times per query as indicated in the general notes section of this chapter. To determine the amount of times we will be querying the model for the answer, we will be querying the model a large number of times as part of this Experiment. When done, we can generate accuracy-of-X scores, which is the accuracy of the Majority-of-X answer from the first X answers against the ground truth. Using these results, we determine a value of X per model assessed, which we will use further in Experiment 2. This parameter is a trade-off between stability of the results, which we will prefer, and speed of inference. This is because we run the inference X times, and as such higher values will result in higher inference times. The only acceptable values of X are uneven. The uneven requirement ensures a majority consensus can be reached at all. We also require the value of X to be higher than 1, as choosing 1 as value of X is the same as just running one stochastically influenced experiment.

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 is a required aspect of the **Majority-of-X** approach, as we need a closed answer set to determine a majority vote. For this proof of concept element, what is the majority is simply the majority vote on the single yes/no answers for a given query.

Lastly, we remark that a yes/no answer differs from the OSADL dataset described in Chapter 3, which also includes the does make a difference between strict and loosely copyleft. In the same chapter, we clarify that this generally does not make a difference and as such we don't need to further specify the answer. We also do not allow the LLM to produce the answer "Questionable", as to ensure it is able to provide an answer every time, rather than defaulting to the neutral answer for the more complex licenses.

4.1.1 Evaluation

We will evaluate the effectiveness of this method using an accuracy score per model for the Majority-of-X answer. We will also discuss some other accuracy scores as well to indicate specific behaviors in Chapter 5.

To do this, we define the following rules for accuracy calculation:

1. We say a model is accurate for a given license if its Majority-of-X answer matches that of the ground truth provided by the OSADL. Otherwise, we say the model is inaccurate.

2. If the OSADL does not provide a ground truth, we do not consider the result in the accuracy score at all.

3. The total accuracy of the model is expressed as the percentage of licenses for which the model is said to be accurate, as per rule 1.

Given these rules, we will be able to assess and rank all models for their accuracy on existing licenses. These results will be discussed in Chapter 5. Based on these results, we are already able to correctly classify a majority of license combinations correctly ($\sim$ 73.276% at 100% accuracy of the model).

4.2 Experiment 2: Combination reasoning

Given the property gained in experiment 1, we can also already attempt to infer the result of two licenses being combined in a particular way. This is because we already observed interesting numbers with respect to the copyleft property in Chapter 3, the most telling of which is the fact that 100% of license combinations where both are permissive are considered to be compatible.

This experiment poses an important question, and its effectiveness is something we will leverage in an attempt to answer the question posed by this thesis: Is the use of large language models in the context of open source software license combinations feasible? Naturally, the answer will not be a simple yes or no.

4.2.1 Approach

To achieve the goal outlined correctly, we recall the following definitions provided by the OSADL:

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

Given these licenses, it is evident that the results of Experiment 1 are closely tied to clauses C2, C3, I2, I3 (of which the explanation is already provided in Chapter 3) and I3. This leaves the clauses C1 and I1 unhandled, but they seem to be more of a formality, as an explicit compatibility clause outside of version upgrades of a given license haven't been observed by this thesis in the dataset. This gives us the ability to define the following decision rules for a given license combination:

1. If the subordinate license can be upgraded to the leading license, **rule incompatible**.
2. If the leading license is permissive ...
   (a) and the subordinate license is permissive, **rule compatible**.
   (b) and the subordinate license is copyleft, **rule incompatible**.
3. If the leading license is copyleft ...
   (a) and the subordinate license is permissive, **rule compatible**.
   (b) and the subordinate license is copyleft, **rule incompatible**. (Explicit compatibility already handled by rule 2).

With respect to upgrade paths, we infer these as direct statements from the license. For example, GPL-2.0-or-later can be upgraded to GPL-3.0-only, and as such, using code licensed under the former license in a project licensed under the latter is a valid combination to make.

4.2.2 Evaluation

We will evaluate the complete experiment via an accuracy score. This accuracy score is simply determined by comparing the end result for every license combination to the ground truth established by the OSADL organisation. However, there is some caveats to this approach:

- In the case that the OSADL indicates a given combination has "Depending compatibility", this means that the combination itself is not compatible, but a given upgrade path is. As upgrade paths are out of scope of the proof of concept presented, we will say a given LLM is accurate if the result is "Incompatible" for these entries.
- If the OSADL does not provide a ground truth answer, indicated by marking the entry as "Unknown", we do not allow answers given here to have an effect on the model's accuracy score. The LLM will have provided a response, but in the absence of a ground truth it is not possible to validate this answer.

Given these caveats and their proposed solutions, an accuracy score can be determined per model and per combination method proposed. We will use these accuracy scores to assess the proposed methods.

Chapter 5

Results

5.1 Experiment 1: Copyleft clause

5.1.1 Determining X for Majority-of-X

As discussed in Chapter 4, we will first determine the value of X for a Majority-of-X answer construction. To do this, we need need to define values for X, based on where we observe the stochasticity to have stabilised. We will do this by making observations up to the value of X = 15, where a requirement for further runs would lower the credibility of the model.

After running the models for multiple times, we can render the following accuracy table of the first 15 Majority-of-X answers for each LLM. It is important to understand that ordering of individual runs is always preserved while calculating this answer. As such, evening out the stochastic behavior is fair and not based on arbitrary decisions.

	Majority-of-X accuracy							
X = ...	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**
deepseek-r1:8b	89.47%	89.47%	92.11%	92.98%	93.86%	93.86%	93.86%	92.98%
gemma3:4b	82.46%	83.33%	83.33%	83.33%	83.33%	83.33%	83.33%	83.33%
llama3	83.33%	85.96%	83.33%	85.09%	85.96%	86.84%	87.72%	86.84%
qwen3:8b	97.37%	96.49%	96.49%	97.37%	97.37%	97.37%	97.37%	97.37%

Table 5.1: Accuracy scores for copyleft detection, rolling value of X for Majority-of-X results

Or in absolute numbers, out of a total of 114 licenses which were given a Copyleft categorization by the OSADL, the following table shows:

	Majority-of-X # accurate results							
X = ...	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**
deepseek-r1:8b	102	102	105	106	107	107	107	106
gemma3:4b	94	95	95	95	95	95	95	95
llama3	95	98	95	97	98	99	100	99
qwen3:8b	111	110	110	111	111	111	111	111

Table 5.2: Absolute number of accurate results for copyleft detection, rolling value of X for Majority-of-X results

Let's now examine these results individually per model.

deepseek-r1:8b: This model appears as the second strongest model for this task in the list. It is also the second most varying model, given that the amount of results it scores correctly varies at 4 points throughout the evolution from Majority-of-1 to Majority-of-15. In terms of absolute change, the most significant change happens at 3 to 5. After this, any change remains within the bounds of 2. At X=7 and onwards, any changes remain within the bounds of 1. For this reason, we choose **7 for the value of X for deepseek-r1:8b**.

gemma3:4b: The gemma3 model appears very stable, already stabilising fully at X = 3. Due to this consistency, we choose **3 for the value of X for gemma3:4b**.

llama3: A varying model which offers great speed and relatively strong accuracy, while among the lower scores in the results. Unlike the other models, this model does not reach a stable point within the first Majority-of-15 result sets. We choose **15 for the value of X for llama3**, as it is a fast model where making multiple runs is not an issue.

qwen3:8b: This model is the strongest on the list, while also having the longest average runtime. We see that for this model, only two points of change occur in the entire table, and this model is therefore very consistent, like gemma3:4b discussed before. Given the stochastic nature of LLM's, choosing value 1 is not appropriate. For this reason, we choose **3 for the value of X for qwen3:8b**.

5.1.2 Accuracy scoring

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

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: 89.10%

Majority-of-7 accuracy: 92.98% (106/114)

OSADL \LLM	Copyleft	Permissive
Copyleft	21	
Permissive	0	85

Table 5.3: Deepseek R1:8b confusion matrix (Majority-of-7)

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 5.1.2, 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: 83.33%

Majority-of-3 accuracy: 83.33% (95/114)

OSADL \LLM	Copyleft	Permissive
Copyleft	10	19
Permissive	0	85

Table 5.4: Gemma3:4b confusion matrix (Majority-of-3)

Llama3: This model performs slightly better than Gemma3:4b, and is also slightly more varied. It provides a fast baseline of what simple models can achieve, allowing you to trade a minor amount of accuracy for faster runtime. However, this model is varied and requires more than 15 runs to become stable, as such, answers provided by this model should be considered as largely stochastic still.

Average accuracy: 83.74%

Majority-of-15 accuracy: 86.84% (99/114)

OSADL \LLM	Copyleft	Permissive
Copyleft	21	15
Permissive	0	78

Table 5.5: Llama3:8b confusion matrix (Majority-of-15)

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, we also observe here that the model actually has a disadvantage when it is placed in a smaller majority configuration like Majority-of-3, as it is susceptible to the first runs that it is fed.

Average accuracy: 96.49%

Majority-of-3 accuracy: 96.49% (110/114)

OSADL \LLM	Copyleft	Permissive
Copyleft	26	4
Permissive	0	84

Table 5.6: Qwen3:8b confusion matrix (Majority-of-3)

5.1.3 General conlusions

These accuracy characteristics display an interesting property: The accuracy of a given model in a majority configuration will continue rising until it stabilises around a given point. This is a logical consequence of the fact that, while an answer for an individual license is deterministic, it will produce an average accuracy for that individual license as more runs are added to the data in consideration. As our accuracy measure is an aggregate of those for individual runs, this measure will display the same property.

It is then a logical conclusion to say that a model will, generally, answer wrongly for a subset of the input licenses. And in this avenue, we examine whether or not we can identify any reasons for these mistakes made by the models. We start by examining the two licenses which all models got consistently wrong.

Universal 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 almost all runs.

- **Sleepycat:** The sleepycat license is also always classified incorrectly. In this case, the source of confusion is a bit more 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.

Individual mistakes

TOFIX ►*Victor: This heading needs a lot more love, and I intend to look at each mistaken license itself, determine where its copyleft clause is or isn't, and based on that, I will explain the model's mistake based on the reasonings it has provided.*◄

Deepseek R1: This model is a well-performing model that only seems to struggle on licenses which other models aside from qwen3 struggle to classify correctly as well. These are the CDDL and MS license families (4 total, though Deepseek specifically does correctly handle the MS-PL license). In terms of mistakes not shared by other models, it seems to struggle on the CPL-1.0 license, as well as both EPL licenses.

Gemma3: We observe two kinds of mistakes for this model. Firstly, for many of the mistakes, we see that this model has answered incorrectly across the board. This makes this model unique in this sense, as none of the other models examined exhibit this kind of absolute behavior. Unfortunately, it is rather hard to determine why it does this. Secondly, we see that it is not as consistent for the MPL license family (3 total on the list) specifically, where it does answer nondeterministically each time the license is examined.

Llama3: The mistakes of this model are harder to discuss in a general sense and appear to be linked to the model's limited reasoning capacity. We observe that in most cases, the model simply picks out an existing clause in the license and then considers that to be the full reasoning for a given decision. Another pattern it provides in its reasonings is a sort of reasoning loop, where it places the confident statement that the license does or does not contain a copyleft clause, and then argues that by providing the definition of such a clause, without making references to the actual license text.

Qwen3: Artistic-2.0, CPL-1.0

5.2 Experiment 2: Combination reasoning

The experiment as outlined in Chapter 4 comes down to simply running an algorithm. When we do this, we obtain the following results table, which has the Majority-of-X parameter in the leftmost column. Results in bold are the results which correspond to the Majority-of-X value chosen for the model at the start of this chapter.

X	Deepseek R1	Gemma3	Llama3	Qwen3
1	86.90% (10706)	79.26% (9765)	80.89% (9966)	94.66% (11662)
3	87.15% (10737)	**80.18% (9878)**	83.98% (10346)	**93.74% (11549)**
5	89.43% (11018)	80.19% (9880)	81.18% (10001)	93.74% (11549)
7	**90.25% (11119)**	80.19% (9880)	82.66% (10184)	94.63% (11658)
9	90.97% (11208)	80.19% (9880)	83.57% (10296)	94.63% (11658)
11	90.97% (11208)	80.19% (9880)	84.25% (10380)	94.63% (11658)
13	90.97% (11208)	80.19% (9880)	85.44% (10526)	94.63% (11658)
15	90.04% (11093)	80.19% (9880)	**84.50% (10411)**	94.63% (11658)

Table 5.7: Combination reasoning results: Perc% (Number)

To make sense of this table, we also list the following facts:

- For 114 licenses, a total of $116^2 = 13456$ permutations exist.
- 116 of these combinations are combining the license with itself, which is always trivially compatible. These were not included in the results table above.
- Of the remaining combinations, the OSADL did not asses 1020. These are also not counted in the results table above.
- The remaining number of combinations that were assessed using the proposed assessment algorithm is: $13456 - 116 - 1020 = 12320$

TOFIX ▶*Victor: This section needs to be finalized by going over the results presented, pointing out interesting facts such as: Which model has performed best, which models are consistent as a direct consequence of a consistent output for experiment 1, biasing the results by only examining results where at least one license is factually copyleft (OSADL) to get rid of the large chunk of Permissive-Permissive licenses, ...*◀

5.3 Answering the ~~posed~~ research questions

TOFIX ▶*Victor: Two to three paragraphs per research question formulating answers to them, basing ourselves off of the conclusions drawn above.*◀

Chapter 6

Conclusion

TOFIX ►*Victor: This remains the template for the thesis, which I will take as instructions to formulate a conclusion chapter.*◄

We recapitulate the thesis by re-examining the context the work is placed in. We address the problem of the mismatch between legal professionals and open source software developers which is clear and evident, and deduce the problem of license understanding based on this key point. It is important to recall that software licenses are still very much necessary in the context of open source software, given that the default of no license imposes restrictions on the software which cannot be reconciled with the goal of open source.

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] Mike Linksvayer. Licensee, a ruby gem to detect under what license a project is distributed. https://github.com/licensee/licensee. Accessed: 2025-06-19.

[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.

[26] Ollama documentation. https://github.com/ollama/ollama/tree/main/docs. Accessed: 2025-05-23.

[27] Llama3 model page on ollama. https://ollama.com/library/llama3. Accessed: 2025-05-24.

[28] Gemma3 model page on ollama. https://ollama.com/library/gemma3. Accessed: 2025-05-24.

[29] Deepseek r1 model page on ollama. https://ollama.com/library/deepseek-r1. Accessed: 2025-05-24.

[30] Qwen3 model page on ollama. https://ollama.com/library/qwen3. Accessed: 2025-05-30.

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.**