

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

Disclaimer Master's thesis

This document is an examination document that has not been corrected for any errors identified.

Without prior written permission of both the supervisor(s) and the author(s), any copying, copying, using or realizing this publication or parts thereof is prohibited. For requests for information regarding the copying and/or use and/or realisation of parts of this publication, please contact to the university at which the author is registered.

Prior written permission from the supervisor(s) is also required for the use for industrial or commercial utility of the (original) methods, products, circuits and programs described in this thesis, and for the submission of this publication for participation in scientific prizes or competitions.

This document is in accordance with the master thesis regulations and the Code of Conduct. It has been reviewed by the supervisor and the attendant.

Contents

0	Preamble	7
	Abstract	7
	Acknowledgments	8
1	Introduction	9
1.1	Authorship rights & open source	9
1.2	Licenses	9
1.2.1	Standardized licenses	9
1.2.2	Custom licenses	10
1.3	Software integration patterns & the definition of related work	10
1.3.1	Application on software	11
1.4	Using LLMs applied to licensing	13
2	Justifying automated license conflict detection	14
2.1	Accepted standards	14
2.2	License scanners (ScanCode, FOSSology)	14
2.3	Custom licenses	14
2.4	License families	14
2.4.1	Copyleft licenses	14
2.4.2	Permissive licenses	14
3	State-of-the-art: OSADL	15
3.1	Open Source License Checklists	15
3.1.1	Format description	15
3.1.2	Defining noteworthy terms	16
3.1.3	Detecting license family based on a checklist	16
3.1.4	16
4	Method / Experimental Design	17
4.1	Important property: Copyleft clause presence	17
4.2	LLM selection	17
4.3	Approach	17
5	Method results	19
5.1	Copyleft clause detection results	19
5.2	Examining mistakes made by models	20
5.2.1	General mistakes	20
5.2.2	Individual mistakes	21
6	Conclusion	22
	References	24

A Relation with Research Projects	25
--	-----------

List of Figures

List of Tables

5.1	Llama3:8b confusion matrix (Run 1/1)	19
5.2	Gemma3:4b confusion matrix (Majority vote of 5 runs)	19
5.3	Deepseek R1:8b confusion matrix (Majority vote of 5 runs)	20
5.4	Qwen3:8b confusion matrix (Majority vote of 5 runs)	20

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, **these people** 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 is 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 Licenses

1.2.1 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 in 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.

1.2.2 Custom licenses

But it is important to remember: Anyone making a work and fixing it becomes the owner of the rights to the work. While choosing a standardized license is possible, one can also write their own license containing terms under which the software can be used.

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 [10]. But it is entirely possible to create one's own license, and some do. The vast number of existing licenses is attested to by the fact that the "Other" bucket of licenses ranks high on the GitHub platform. This is the bucket which contains all licenses that can't be identified by scanners, because it isn't widely used enough to be known [7].

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 [11][12][13].

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 is 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 remain 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.**

This aligns with the definition of "derivative works" in U.S. Copyright Law [1][14][15][16].

- **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][16].

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 Using LLMs applied to licensing

When we consider the concept of licensing however, we must understand that we are still dealing with natural language. This means we can 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.

The option this thesis examines is using LLMs (large language models), with OpenAI's Chat-GPT being a primary inspiration, to discern the necessary properties to advise any person about specific license combinations in the full-inclusion pattern.



Chapter 2

Justifying automated license conflict detection

This thesis examines a problem and proposes a solution to it within the field of open source software licensing. This is the practice of

2.1 Accepted standards

SPDX, ReUse Project,
OSADL (short-form)

2.2 License scanners (ScanCode, FOSSology)

ScanCode
FOSSology
Indicate the fact that these work based on a pre-defined matrix, or don't identify conflicts at all.

2.3 Custom licenses

Indicate that custom licenses (licenses only one project uses) are in use in the open source landscape, reason why, demonstrate how

How in two ways: - GitHub innovation graph placement of "Other" category - Individualised licenses in dataset

2.4 License families

2.4.1 Copyleft licenses


2.4.2 Permissive licenses

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 [17].

Key Objectives and Activities of OSADL:

- 
1. **Real-time Linux Development:** OSADL plays a major role in maintaining and enhancing the **PREEMPT-RT patchset**, which enables deterministic real-time performance in the Linux kernel. This is critical for automation and embedded systems where precise timing is required [17].
 2. **Legal and License Compliance:** OSADL helps member companies ensure their use of open-source software complies with licensing obligations. This includes providing tools and services for legal audits and documentation [17].
 3. **Testing and Quality Assurance:** OSADL maintains testing infrastructure such as the OSADL QA Farm, which continuously tests real-time and mainline Linux kernels on a wide range of hardware to ensure stability and performance [17].
 4. **Community and Standardization Engagement:** OSADL represents the interests of industrial users in various open-source communities and standardization bodies, helping to align industry needs with open-source development [17].
 5. **Member Collaboration:** As a consortium, OSADL is funded and driven by its members, which include automation vendors, silicon manufacturers, software developers, and end users. It fosters collaboration among these stakeholders [17].

It is natural that this thesis primarily will examine the second item in this list.

3.1 Open Source License Checklists



The OSADL and its members allow the examination of specific licenses for the creation of a license checklist. **Th**

3.1.1 Format description



Describe the main body of the format. Also describe the annex section of the format and point out that this is manual work.

3.1.2 Defining noteworthy terms

Touch on copyleft and other very important terms here.

3.1.3 Detecting license family based on a checklist

3.1.4

TOFIX ► This needs further work. ◀



Chapter 4

Method / Experimental Design


4.1 Important property: Copyleft clause presence

TOFIX ▶ Describe why this is so important, calling back to the OSADL dataset where this property can already be observed. ◀

4.2 LLM selection

TOFIX ▶ Describe what OLLama is. Describe why each model was chosen. ◀

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 [18][19].

Lastly we remark that this query poses a yes/no question which limits 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 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% (102.8 / 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% ($\tilde{111.6}$ / 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 preceded 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>.
- [5] Spdx license list. <https://spdx.org/licenses/>.
- [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.
- [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] 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.
- [11] 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.
- [12] Licensing issue on the tuxedocomputers/tuxedo-drivers gitlab repository. <https://gitlab.com/tuxedocomputers/development/packages/tuxedo-drivers/-/issues/137>. Accessed: 2024-11-27.
- [13] Licensing issue on the archived tuxedocomputers/tuxedo-keyboard github repository. <https://github.com/tuxedocomputers/tuxedo-keyboard/issues/61>. Accessed: 2024-11-27.
- [14] 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.

- [15] 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.
- [16] 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.
- [17] Open source automation development lab (osadl) eg. Accessed: 2025-04-13.
- [18] Scancode documentation. <https://scancode-toolkit.readthedocs.io/en/stable/getting-started/home.html>. Accessed: 2024-12-18.
- [19] Fossology homepage. Accessed: 2024-12-18.

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.**