Computer-aided conflict detection between open-source software licenses

Victor van Herel
Student Master of Computer Science
University of Antwerp
victor.vanherel@student.uantwerpen.be

I. Introduction

This project was made in the context of a Master's thesis project for the course Computer Science at the University of Antwerp. It is promoted and supervised by Prof. Dr. Demeyer, and throughout writing the author was mentored by Mutlu Beyazit.

~~*A. Problem statement*~~

Software licensing in the open source landscape is a tool for software authors and repository maintainers to keep control over how their contributions are used [1]. To this end, a large number of open source software licenses exist which are approved by the Open Source Initiative, which maintains a list of licenses which are compliant with the Open Source Definition, maintained by the same organization [2] . However, outside of this license, more exist, as while it is common for repository maintainers to choose an existing license that fits their needs, there is nothing stopping someone from writing their own license.

Software licenses are complex entities however, as they describe a set of permissions and prohibitions in legal terms, which often do not necessarily map trivially to the actions which these licenses intend to govern in practice. This mismatch has lead to a situation in which repositories exist that are knowingly or unknowingly violating software licenses in their repositories. [3]

The primary reason behind the initial proposal that drove this project is the fact that it is very easy to come into contact with software licensing in the open source landscape as an aspiring young contributor, who often do not have a legal background necessary to understand the full text of a complexly built license.

This project aims to develop part of a solution that allows uneducated members of the open source community to understand how software licenses are built up and work.

The author is additionally aware of tools that already exist in the landscape such as ScanCode and Fossology which are used to scan software repositories for the licenses they use, either implicitly or explicitly. Such tools can also benefit from this work, by virtue of the fact that this can automate the conflict detection between detected licenses, especially in the case of custom licenses. It is our understanding that these tools work based on a predetermined matrix of licenses and their compatibility, which we will use as a baseline for our work [4], [5] .

B. Research statement

The majority of the research presented in this paper focuses on applying new developments in the field of artificial intelligence to the subject of software licensing. For this, the author has chosen to employ large language models to determine key properties of any arbitrary software license text.

The project is a feasibility study to determine the effectiveness of these advances in answering a distinct set of questions. As such, we define the following areas of research:

- **RQ1:** Which software interaction patterns exist that are relevant to software licensing problems?

  To define compatibility and incompatibility between two licenses, we need to frame this in a context in which the (in)compatibility occurs. These contexts specifically concern a difference in the amount of license provisions that apply to a composite work from the combined license package, based on the way that it was composed in.
- **RQ2:** How do we model a software license into a machine-readable format?

  This research question focuses on the definition of a machine-readable format which can be used to represent the provisions made in a license. The purpose behind this is to be able to have an intelligent system provide clear reasoning behind a decision, after the examined licenses have been distilled into such a format.
- **RQ3:** How can we deploy large language models to ascertain specific characteristics of a license, with the goal of distilling licenses into the format determined by RQ2?

  ~~This is the capstone element to this project.~~ When we are able to automatically distill licenses into a well-defined machine readable format accurately, it is possible to automatically interpret any arbitrary software license text and test it against others.

This study aims to answer the feasibility of the questions posed, providing a proposal for a solution where able, or showing what needs to be done where a solution is not immediately possible.

II. Existing tools used

A. Language concepts and elements

To be able to define a coherent schema into which we can represent software licenses, we need to define language terms with a coherent and singular meaning. These defined terms will help us to construct a framework with which we can define permissions, obligations and other clauses imposed by licenses, and thus represent a license in full.

To this end, the Open Source Automation Development Lab (henceforth OSADL) organization has already provided a lot of work in the form of their License Checklists, which provide a baseline for the work proposed here [6]. Specifically, the checklist documents are documents which are tailor-made for specific situations OSADL's members propose. However, the raw data that backs these checklists is made publicly available via their website, and this data can be programmatically downloaded as well. It is this raw data, and the language elements that compose it, which are useful to us in particular.

The OSADL organization categorises their language constructs in three categories[6]:

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

A full overview of these terms, along with their definitions, can be reviewed in Annex B .

Example: The simplest example provided is the raw representation of the MIT license (which as a refresher includes the permission to do virtually everything with the source code, on the condition that the license text is included, and that no warranty is claimed[7]), which shoes how these language constructs work together to convey the meaning of a license:

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

It should be noted that indentation is also used in this language to define a subordinate instruction to the primary instruction. Additionally, usually, instructions on the top level are USE CASE instructions, which allow licenses to provide separate provisions for separate use cases which may be relevant to licensing.

B. Large language models

There is many large language models available via the Ollama tool, which can be used for running large language models locally [8]. This is the approach we will be taking during this study. Details of the replication package can be found in .

For this study, we selected four of the best performing models which are published publically. The reasoning behind these choices are as follows:

- **Llama3:8b :** Llama3, by Meta, advertises itself as a strongly capable model and is relatively fast. It is not a thinking model, which means it won't enter into an internal monologue first [9]. Other selected models do have this feature, which we will see will perform much better on this complex task [10].
- **Gemma3:4b :** A more recently published model by Google, which is listed to be more powerful than the Llama3 model, at the cost of increased runtime. This model is not a thinking model [11].
- **Deepseek R1:8b :** This model has been around for a while and is the model that caused the LLM space to transform significantly in its wake [12], [13] . This model is a thinking model which empowers it to perform more complex reasoning tasks [9]. This approach does come at a cost of longer inference times, which we will not consider for this study as long as it does not make the work package infeasible [14].
- **Qwen3:8b :** The most recent model of this list. Qwen3 is also a thinking model. This specific model has a Mixture of Experts architecture, which is another empowering factor for this model [15]. Of all models presented, this model should range similarly with Deepseek R1 in complexity, though its documentation shows it is capable of outclassing Deepseek R1 in general benchmarks [16].

We will be performing the main tasks of this study with the selected models, running them multiple times where able to assess average performance rather than a one-shot lucky inference. This is required due to the fundamentally stochastic nature of all large language models.

III. License-relevant software interaction patterns

The answer to RQ1, the question that examines the possible ways in which software can interact relevant to the world of software licensing builds forward upon patterns found in the software licenses themselves.

We would like to re-introduce the concept of a Copyleft license family, and a Permissive license family.

- **Copyleft:** A license that belongs in this family usually 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. It is also possible that such a license has a loosely copyleft requirement, requiring that the provisions in the license (such as requirement to display the original license, …) are abided by, and that the derivative work is then distributed under a *compatible* license. A compatible license in this sense is a license that is usually explicitly called out, or vaguely described as imposing at least the same restrictions.

  The point of copyleft licenses is to ensure the openness of the work by use of a license. Copyleft licenses almost always require an author to make the work open source, and due to the reciprocity effect, any derivative works must also be made open source [17], [18].

  Examples of such licenses are GPL, LGPL, …

- **Permissive:** 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, but is a very necessary tool for work authors as adding no license at all essentially disallows any use at all.

  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 [17], [18]. In fact, within the OSADL dataset, there are no permissive licenses that are listed as incompatible with another permissive license. This can be referenced further in Annex C.

  The most known permissive license is the MIT license. Other examples include the BSD licenses, the Unlicense, …

~~The identification of these software license families based on the software license text is an interesting property to determine first for the purposes of RQ3, as the correct identification of a family to which a license belongs allows us to quite easily determine compatibilities with other licenses which match the same family in the case of a derivative work.~~

In terms of licensing conflicts, most conflicts only arise from this requirement that a derivative work be similarly licensed. To tackle this in formal terms, we must first define the following:

- **Derivative work:** The United States Copyright Act defines this as "a work based upon one or more preexisting works […] in which a work may be recast, transformed or adapted" [19]. This is a generalized definition that applies to all domains where copyright can apply, not just coding. In practice for code, a derivative work is a code base, project or otherwise collection of code that uses some code from another, potentially differently licensed source.

  Cherry-picking commits among works creates derivative works. So does taking an archived repository and beginning to actively maintain it. Generally, whenever you take a section of code, however small, from another source and use it in your own source code, you create a derivative work. In low-level languages, using a a static library also includes the result of that source code into the binary distribution, which means the work is also a purely derivative work [17], [19].

  Licensing conflicts arise here when the leading license (the license of the project itself) is not compliant with all instructions provided in the subordinate license (the license under which the other work is covered). A clear example: An MIT (Permissive) licensed repository can not guarantee that it is able to keep code open-source, as the license does not contain such a provision. It is therefore not able to use parts of a GPL (Copyleft) licensed repository in its own source code, as this license which covers the included parts requires that those parts remain open source in all variations. It should be noted here that we are referencing a full integration of the shared code snippets into the leading code base, thus separation between the coverage of each license in play is not clear.
- **Collective work:** The United States Copyright Act defines a collective work as "a work […] in which a number of contributions, constituting separate and independent works in themselves are assembled into a collective whole" [19] . Again, this definition is generalized, and for the programming domain where software licensing applies, this usually applies to works which use libraries. The leading license is still the license governing the work that is being created, but each library used may be governed under its own license. It is important that code, or binary derivations of it, of the library does not end up in the distributed work. The interaction can be defined rather freely however, dynamic linking for low level languages, python dependencies, java classpath inclusions, … all apply in this case.

  Generally, unless a collective work is also a derivative work, subordinate licenses do not impose restrictions on work done in a leading context. It should be noted that licenses may carry their own definition of what a derivative work truly is, as is the case with the GPL license which expands the definition of a derivative work to include works that use any GPL code even via dynamic linking. The LGPL license is a modified version of the GPL license which excludes this clause.

To conclude, from the perspective of the software license world, it matters a lot whether or not a derivative work is created. If this is the case, license restrictions of the original work fully apply. These cases are the further focus of this study, and we keep rulings on collective work interactions out of scope.

IV. Machine-readable software license format

A. *Modeling a license*

To define a format that is able to represent licenses requires us to examine what a license actually is. For this, a helpful model has been proposed that regards a license L as a set of grants of rights. Additionally, conditions for a grant to a right r is a set of conjunctive assertions. All conjuncts should be satisfied for the licensor to receive such a grant [19].

This is a useful mental model to keep in mind when handling a license. For example, the cited work [19] includes the example of the BSD 4-clause license which grants the recipient the right to distribute derivative works in binary form, provided the following conditions are adhered to:

1) "Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution."
2) "All advertising materials mentioning features or use of this software must display the following acknowledge-

ment: This product includes software developed by the <copyright holder>."

3) "Neither the name of the <copyright holder> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission."[19]

B. *Work by OSADL*

The Open Source Automation Development Lab organization already uses a specific format for summarizing licenses. Their approach differs from the grant-oriented approach proposed above, and orients itself around use cases. A number of use cases are defined, which each orient itself around the delivery of a particular object. We name two examples, along with the definition provided by the OSADL organization for the term.

- 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.

These use cases map directly to the context in which we are creating a derivative work, and orient themselves about what is being delivered. Do note that is is possible that use cases are combined into one logical block. This would look like the following: `USE CASE delivery_a OR delivery_b OR ...` for multiple types of delivery use cases which are subject to the same obligations and prohibitions that are listed under the `USE CASE` line.

Indented under `USE CASE` lines, obligations (e.g. `YOU MUST`) and prohibitions (e.g. `YOU MUST NOT`) are listed in plain language, however, the objects of these statements are again based on defined terms only. These terms are included in Annex B.

Lastly, after the `USE CASE` descriptions which contain the main body of the license, the organization defines that on separate lines, any number of the following statements may be included:

- `COMPATIBILITY license_id`
- `INCOMPATIBILITY license_id`

This is used for defining explicit compatibilities and incompatibilities which are included in a license, but in OSADL's provided dataset, often a lot more of these statements are present as opposed to what the license actually rules as explicitly compatible or incompatible.

Working with either format is very complex, due to the sheer number of terms that exist. This is inherent to the nature of software licensing however, and as such a simpler format will generally make omissions.

The license format used is however able to be read and interpreted by machines, provided it was constructed correctly, as the format is rigid in terms of what it is allowed to contain, and as such it is not infinite. This study intended to show this fact based on the baseline of the model, but was not able to due to time constraints, and provides it as future work.

C. *OSADL's definitions of compatibility and incompatibility*

It is noteworthy to specifically analyze OSADL's definitions of compatibility and incompatibility. These definitions are provided verbatim:

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

A noteworthy mention is that these definitions define exactly two classes disjoint classes. This is easy to see when we consider each possibility of copyleft separately:

- **Both copyleft:** Either covered by compatibility clause 1 or incompatibility clause 3 based on the presence of an explicit compatibility clause in either license.
- **Leading copyleft, subordinate permissive:** Compatible by clause 3 of compatibility, or incompatible by clause 2 of incompatibility based on the presence of an incompatible obligation.
- **Leading permissive, subordinate copyleft:** Incompatible by clause 2 of incompatibility, given that copyleft licenses impose restrictions that permissive licenses do not keep, unless explicitly compatible which is covered by clause 1 of compatibility.
- **Both permissive:** If both are permissive, compatibility is ruled by clause 2 of compatibility.

This is an important property to show as a license should only be able to to be compatible, or to be incompatible, and not both at the same time or neither in a well-defined system.

When we examine the OSADL-provided dataset, we find there is indeed no license combinations in the whole matrix containing 116 licenses which is both, however, there do exist a number of combinations where no answer is provided. (Compatibility value listed as UNKNOWN.)

V. Discerning properties of a license with large language models

This section of the study goes deeper on the implementation of the facts we have already outlined, and may be reproduced via the reproduction package.

A. General data elements

For this part of the study, a number of data elements was required. We rely upon the following:

- The SPDX license index for providing full text versions of license we examine. This is provided in the GitHub repository spdx/license-list-data.
- The OSADL dataset for baseline values described in Annex C.

The reproduction package contains descriptions of how to access both.

B. Determining presence of a copyleft clause

A first question which this study aims to answer is whether or not large language models can be applied to accurately determine whether or not the license belongs to the Copyleft family of licenses, or the Permissive family of licenses.

Discerning this data point is useful as it allows us to immediately known if license incompatibility for a given subordinate license is a concern at all. In fact, in the dataset, the only situations in which the OSADL organization detects a conflict between two licenses where the subordinate license does not have a copyleft clause (10005 applicable combinations), only 248 combinations are explicitly reported as incompatible. Rather unhelpfully, the organization's explanation of this decision always relies on an explicit `INCOMPATIBILITY` statement in their distilled license, with no provided reasoning behind the presence of that mention.

Experiment method: [illegible] all 116 licenses, the selected large language model is provided the full text of the license, and is asked whether or not the license contains a copyleft clause. If it does, it is ruled as a copyleft license, if it doesn't, the converse happens.

Given this description, the LLM is essentially tasked with classifying the licenses into the categories defined. To eliminate the variable of stochastic LLM behavior, the final result to be analysed is a majority vote of 5 separate runs of the same model.

Results:

- Llama3:8n: Upon a first run of this model with the given task, it was decided not to continue with the additional runs for this model, as the provided accuracy was very low and as such Llama3:8b is not a promising avenue. The confusion matrix for this run is provided in Table I. 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 I: Llama3:8b confusion matrix run 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 II, 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 II: 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 III: 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% (~ 111.6 / 116)
  Majority-of-5 accuracy: 96.55% (112 / 116)

TABLE IV: Qwen3:8b confusion matrix (Majority vote of 5 runs)

LLM \ OSADL	Copyleft	Permissive
Copyleft	26	1
Permissive	3	86

Examination of mistakes made:

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.

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

Other mistakes do not appear to form a coherent pattern. Additionally, among the list of most used licenses according to GitHub’s Innovation Graph project, mistakes still get made by Gemma3 and Deepseek R1 within this subset. Qwen3 remains faultless within this subset across all of the recorded runs [20].

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.

C. *Distilling the license to the OSADL license format*

This is a much harder problem to tackle, and this study has not been able to show that this is possible in any of the tested models. The problem arises from the fact that it is easy to get a model to answer with a specific word in the beginning for automatic detection. Is becomes more challenging when it must contain its entire answer within a given schema, which becomes even harder as the complexity of said schema increases.

As such, given the significant complexity of the OSADL provided schema, this section of the study is fruitless.

We include as future work the definition of a simpler to understand and to write schema, as LLMs have been shown to be able to answer in specific schemas through prompt engineering. Some models even provide first-class support [21].

D. Naive approach on a limited subset of the dataset

Experiment: For the limited subset of 24 most used licenses[20] that are also in our dataset which we have a baseline for, the number of combinations that cannot be trivially resolved is low enough such that querying a model for each unknown combination becomes feasible, provided we also do not evaluate combinations that OSADL has indicated no baseline answer is available for.

This prompt simply consists of the full text of both licenses, clearly indicating which is the leading license, and which is the subordinate license, and asking it to evaluate the definition of Compatibility and Incompatibility, which is also included in the prompt. The instruction then simply asks to answer Compatible, Incompatible, Both or Neither.

This study has run this approach for the Gemma3, Deepseek R1 and Qwen3 models.

TODO: **Results:** *Pending inclusion. These experiments are still being run.*

TODO: **Examination of mistakes made:** *Pending inclusion. These experiments are still being run.*

TODO: **Impact of this finding:** *Pending inclusion. These experiments are still being run.*

VI. Concluding remarks

Pending inclusion based on currently absent results.

References

[1] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. M. German, and D. Poshyvanyk, “When and why developers adopt and change software licenses,” in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 31–40. doi: 10.1109/ICSM.2015.7332449.

[2] “Licenses approved by the Open Source Initiative.” Accessed: May 21, 2025. [Online]. Available: https://opensource.org/licenses

[3] A. Mathur, H. Choudhary, P. Vashist, W. Thies, and S. Thilagam, “An Empirical Study of License Violations in Open Source Projects,” in *2012 35th Annual IEEE Software Engineering Workshop*, 2012, pp. 168–176. doi: 10.1109/SEW.2012.24.

[4] “ScanCode documentation.” Accessed: Dec. 18, 2024. [Online]. Available: https://scancode-toolkit.readthedocs.io/en/stable/getting-started/home.html

[5] “Fossology homepage.” Accessed: Dec. 18, 2024. [Online]. Available: https://www.fossology.org/features

[6] “OSALD Open Source License Obligation Checklists homepage.” Accessed: Apr. 13, 2025. [Online]. Available: https://www.osadl.org/OSADL-Open-Source-License-Checklists.oss-compliance-lists.0.html

[7] “MIT License text.” Accessed: Nov. 22, 2024. [Online]. Available: https://opensource.org/license/mit

[8] “Ollama documentation.” Accessed: May 23, 2025. [Online]. Available: https://github.com/ollama/ollama/tree/main/docs

[9] T. Wu, J. Lan, W. Yuan, J. Jiao, J. Weston, and S. Sukhbaatar, “Thinking LLMs: General Instruction Following with Thought Generation.” [Online]. Available: https://arxiv.org/abs/2410.10630

[10] “LLama3 model page on Ollama.” Accessed: May 24, 2025. [Online]. Available: https://ollama.com/library/llama3

[11] “Gemma3 model page on Ollama.” Accessed: May 24, 2025. [Online]. Available: https://ollama.com/library/gemma3

[12] DeepSeek-AI *et al.*, “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.” [Online]. Available: https://arxiv.org/abs/2501.12948

[13] N. S. Chong, “Deepseek R1: Pioneering Open-Source 'Thinking Model' and Its Impact on the LLM Landscape.” Accessed: May 23, 2025. [Online]. Available: https://c3.unu.edu/blog/deepseek-r1-pioneering-open-source-thinking-model-and-its-impact-on-the-llm-landscape

[14] “Deepseek R1 model page on Ollama.” Accessed: May 24, 2025. [Online]. Available: https://ollama.com/library/deepseek-r1

[15] O. Sanseverio, L. Tunstall, P. Schmid, S. Mangrulkar, Y. B., and P. Cuenca, “Mixture of Experts Explained.” Accessed: May 30, 2025. [Online]. Available: https://huggingface.co/blog/moe

[16] “Qwen3 model page on Ollama.” Accessed: May 30, 2025. [Online]. Available: https://ollama.com/library/qwen3

[17] H. Kaminski and M. Perry, “Open Source Software Licensing Patterns.” Accessed: Oct. 21, 2024. [Online]. Available: https://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=1009&context=csdpub

[18] R. Sen, C. Subramaniam, and M. L. N. and, “Determinants of the Choice of Open Source Software License,” *Journal of Management Information Systems*, vol. 25, no. 3, pp. 207–240, 2008, doi: 10.2753/MIS0742-1222250306.

[19] D. M. German and A. E. Hassan, “License integration patterns: Addressing license mismatches in component-based development,” in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 188–198. doi: 10.1109/ICSE.2009.5070520.

[20] GitHub, “GitHub Innovation Graph.” [Online]. Available: https://github.com/github/innovationgraph

[21] “How to return structured data from a model | Langchain.” Accessed: Jun. 03, 2025. [Online]. Available: https://python.langchain.com/docs/how_to/structured_output/

[22] “OSADL license compatibility matrix application.” Accessed: Oct. 23, 2024. [Online]. Available: https://www.osadl.org/html/CompatMatrix-noexpl.html

I. Annex A: Replication package

A. Scripts

The replication package is a python project that contains numerous scripts that run the experiments defined, as well as analysis scripts. An overview is provided:

- `copyleft_clause.py`: This script generates one run of the copyleft clause detection experiment for a model provided in its top-level variables. Results are stored in `{model_name}_copyleft.json`.
- `evaluate_subset.py`: This script evaluates all possible combinations of a subset of the licenses provided in its top-level variables. It assesses each combination, and allows (configurable via top-level variable) passthrough to an LLM query for non-trivial decisions. Results are stored in `{model_name}_matrix.json`.

B. Queries

TODO: Documentation of final queries used will be included here.

II. Annex B: OSADL terms

TODO: Include a full table of all terms, and a short description of each category.

For now, the terms can be referenced here:

Language elements: https://www.osadl.org/fileadmin/checklists/all/language.txt
Actions: https://www.osadl.org/fileadmin/checklists/all/actions.txt
Terms: https://www.osadl.org/fileadmin/checklists/all/terms.txt

III. Annex C: OSADL dataset description

The OSADL dataset consists of a number of items:

- License raw checklists: These are the distilled versions of the licenses in the format described in Section IV.B. The organization has provided this for 116 different licenses.
- Language elements: These are described already in Annex B.
- A matrix that defines incompatibility or compatibility between a leading and subordinate license. This file can be examined interactively in the cited material.[22]
  - This matrix covers 13456 possible combinations, of which for each combination, an answer is given:
    - Same, in the case both licenses are the same. (116)
    - Yes, in the case the combination is compatible. (8812)
    - No, in the case it is not. (3448)
    - Unknown, in the case OSADL has not examined this specific combination. (1020)
    - Check dependency, in the case compatibility is dependent on the upgrade of a license to a newer version which is compatible, where the existing license is not. (60)