Redefining Open-Source Compliance: Advanced Information Retrieval for License Detection at Scale

Anonymous ACL submission

Abstract

The integration of Open Source Software (OSS) components in modern software has driven innovation but also highlighted the need for robust license compliance solutions. Given the diversity of OSS licenses, compliance poses challenges for organizations balancing innovation with legal obligations. This paper presents a license scanning tool that accurately identifies and analyzes OSS license statements within projects using Natural Language Processing (NLP) and information retrieval techniques. By implementing four algorithms and advanced similarity measures, our approach enhances license detection accuracy. We further incorporate preprocessing strategies for improved text analysis precision. Future work involves machine learning integration and dataset expansion to elevate compliance assurance. This research not only advances OSS license compliance but also lays groundwork for future improvements via machine learning and community collaboration.

Keywords: license compliance, open source license, software license, information retrieval, information extraction

1 Introduction

Open-source software (OSS) promotes free distribution, allowing access, modification, and redistribution within limits set by creators. OSS licenses protect these rights, specifying usage restrictions to balance openness with intellectual property protection. Verified by entities like the Open Source Initiative (OSI), popular licenses include Apache, MIT, Mozilla Public License, BSD, and GPL.

As OSS adoption grows, effective license management becomes essential. Standardized licenses, like those in the SPDX® License List, streamline compliance, but reviewing and managing licenses across large codebases with thousands of files is challenging. Compliance with OSS licenses is not solely a technical concern but a nuanced process that requires understanding community norms and managing organizational risk (Wintersgill et al., 2024). OSS code's adaptability across projects often leads to accidental license conflicts, highlighting the need for automated scanning tools for accurate identification.

Despite standards like the REUSE Specification (Free Software Foundation, 2019), compliance remains inconsistent. Developers often use license snippets or headers that only partially reflect full license texts, complicating identification—especially when the full license resides in separate files like "LICENSE.ext" or "COPYING.ext."

Advanced license detection algorithms, such as information retrieval techniques, textual similarity, and machine learning, address these challenges. Unlike basic rule-based methods, which often rely on regular expressions, these advanced approaches handle varied and partial texts with improved accuracy. This paper explores various detection algorithms, including Term Frequency-Inverse Document Frequency (TF-IDF) (Sparck Jones, 1972), Damerau-Levenshtein Distance (DLD) (Levenshtein, 1966), N-Gram Cosine Similarity, and word frequency similarity, as shown in Figure 1. These methods each bring unique strengths, from capturing term significance to evaluating text similarity based on minimal edits.

While rule-based systems remain common, they often struggle with subtle license variations. By leveraging advanced information retrieval and textual similarity techniques, our methods aim to reliably detect licenses even when texts are fragmented or incomplete.

2 Related Works

This section outlines previous research and existing work in the area of open-source license compliance, which has significantly influenced the approach

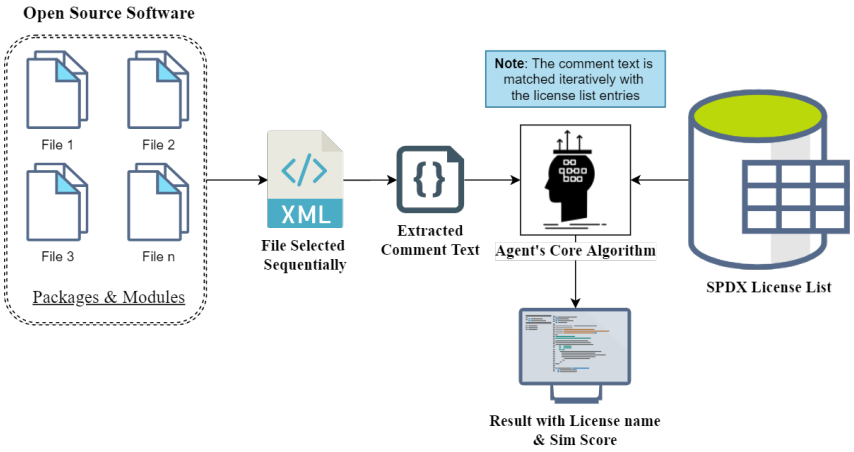

Figure 1: Flow of the License Scanning Process

developed in this paper.

Vendome et al. examined the challenges posed by incompatibilities between different open-source licenses, particularly in projects where multiple licenses are used simultaneously (Vendome et al., 2017). Their work highlighted how such license conflicts can create complications for developers and introduce legal risks during the software development process. Given the widespread use of open-source components across various software ecosystems, open-source license compliance has become a critical issue, as emphasized by Schöttle in his work on the importance of license compliance in modern software development (Schoettle, 2019).

Zhang provided a comprehensive analysis of existing tools for open-source license compliance, comparing the effectiveness of several software solutions designed to identify and manage license obligations (Zhang, 2020). His detailed assessment of various tools highlights the need for robust solutions capable of handling the complexities of open-source software. Similarly, Jaeger et al. discussed one such tool that has been widely used in the industry, which performs license scanning and generates compliance reports based on license and copyright detection (Jaeger et al., 2018).

Lerner et al. explored the broader context of open-source licensing, delving into the complexities involved in selecting appropriate licenses for open-source projects and the potential pitfalls that developers must navigate when doing so (Lerner, 2005). Gobeille also provided an in-depth look into one of the most widely adopted license scanning tools, offering insights into its architecture, usage, and the open-source license landscape more generally (Gobeille, 2008).

The work of Sen et al. focuses on the decision-making process behind the selection of open-source licenses, presenting factors that influence license choice and providing recommendations for developers on how to manage license obligations within their projects (Sen et al., 2008). These previous studies collectively demonstrate the critical need for effective open-source license management and compliance in today's software development environment, where code reuse and collaboration are increasingly common.

3 Methodology

The framework presented in this paper is designed to allow users to choose the algorithm for license scanning, with some algorithms offering the flexibility of different vectorizing and similarity measures. The core functionality revolves around the input file to be scanned, and this section discusses the workflow and implementation of various matching algorithms.

3.1 Preprocessing

Typically, open-source projects provide a `LICENSE.txt` file in the base directory, but additional license texts are often embedded directly within source code files. These license texts, commonly in the form of comments, can differ from the main license file, necessitating sophisticated license scanning approaches. License texts might appear anywhere within the file, interspersed with other non-relevant comments, which complicates the extraction of the actual license statement.

For comment extraction, the system uses a

Python library that scans the entire source code for comments, returning an object consisting of all comments, including potential license text. From this output, license-specific comments—those containing keywords like 'source', 'free', 'under', 'use', 'copyright', 'grant', 'software', and 'license'—are filtered for further scanning.

The system matches the identified text against SPDX licenses to determine which license(s) may apply. During installation, the system fetches the SPDX License List and exceptions from the SPDX API, storing the data as a CSV file. If the latest version is already available, the download is skipped.

The preprocessing steps applied to both the downloaded SPDX license text and the identified license comment include:

- Lower-casing all characters.
- Converting "copyright" or "(c)" to "copyright" to treat both as equivalent.
- Removing all punctuation.
- Removing exceptional characters to simplify and normalize the text.

These preprocessing steps ensure that the texts are in a comparable format, aiding in the application of similarity algorithms. The system also creates an N-gram knowledge base from the SPDX license list, described in Section 3.6.

3.2 Cosine Similarity

Cosine similarity is a widely used measure in information retrieval and text mining, which calculates the similarity between two documents or text fragments by measuring the cosine of the angle between their respective vectors in a multi-dimensional space. The vectors are constructed from the terms in the documents, and the cosine of the angle between them reflects how similar the documents are to one another. A cosine similarity score of 1 indicates perfect similarity, while a score of 0 signifies no similarity at all (Salton et al., 1975; Singhal and others, 2001).

In this approach, each document (or token) is represented as a vector, and the mathematical formula for cosine similarity is derived as follows:

$$\text{LT1} \cdot LT2 = |LT1| \times |LT2| \cos\theta$$

$$\text{Similarity} = \cos\theta = \frac{LT1 \cdot LT2}{|LT1| \times |LT2|}$$

$$\cos\theta = \frac{\sum_{i=1}^{n} LT1_i \cdot LT2_i}{\sqrt{\sum_{i=1}^{n} LT1_i^2} \times \sqrt{\sum_{i=1}^{n} LT2_i^2}}$$

where:

- $LT1$: License Text 1,
- $LT2$: License Text 2.

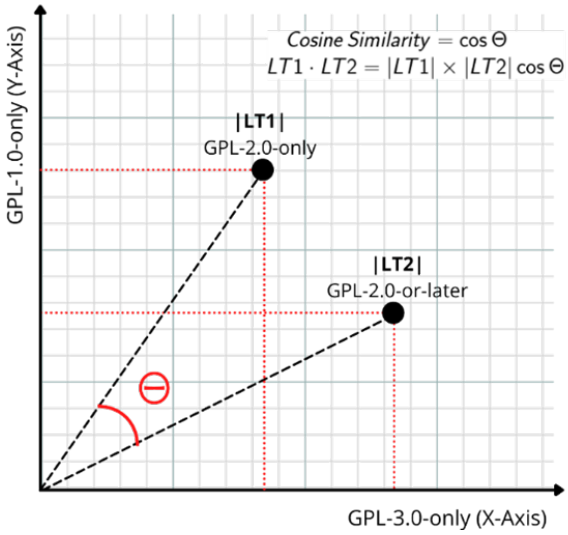

Figure 2: Cosine similarity between two license texts.

One of the major strengths of cosine similarity is that it remains unaffected by the size of the documents being compared, making it a highly useful and normalized metric, with values always ranging between 0 and 1. In the context of license text detection, cosine similarity helps efficiently compare texts by treating the documents as vectors and finding how closely aligned they are.

This similarity measure is employed as the default metric in the system, which uses it to compare the vectors generated by different agents. For instance, in the TF-IDF agent, cosine similarity is used to compute how similar the vectors created through the TF-IDF algorithm are to vectors of other license texts. This allows the system to efficiently identify and rank similar licenses based on their similarity score.

Algorithm 1 Cosine Similarity Pseudocode

```
p ← 0
for each (key, count) in a.items() do
    if key in b then
        p ← p + b[key] * count
    end if
end for
t ← √(a · a) * √(b · b)
if t = 0 then
    return 0
else
    return p/t
end if
```

In this pseudocode, the process is described step-by-step, where two vectors (a and b) are compared by computing the dot product and then normalizing this result by dividing by the product of the magnitudes of the vectors. This final value represents the cosine similarity between the two vectors, which is used to assess the similarity between the corresponding license texts.

3.3 TF-IDF Agent

Term Frequency-Inverse Document Frequency (TF-IDF) (Sparck Jones, 1972) is a well-established information retrieval algorithm that has been widely adopted across various industries due to its simplicity and effectiveness in extracting relevant information from large corpora. The main utility of TF-IDF lies in its ability to assess the importance of terms within a document relative to a larger set of documents. This relevance is computed by measuring the frequency of a term in a specific document against its occurrence across the entire document set, allowing us to weigh the uniqueness and importance of a term in conveying the overall message of the document.

The TF component of TF-IDF stands for Term Frequency, which refers to how frequently a specific word or term appears within a document. However, using TF alone can bias the results toward commonly used terms, such as stop-words like "is," "the," or "and," which do not contribute significantly to the meaning of the document. To counter this, the IDF component—Inverse Document Frequency—is introduced. The IDF measures how common or rare a term is across the entire document corpus, with the underlying assumption that rarer terms carry more significant information than common ones.

Mathematically, the Term Frequency (TF) is calculated by dividing the frequency of a term by the total number of terms in the document. On the other hand, the Inverse Document Frequency (IDF) is determined by dividing the total number of documents in the corpus by the number of documents that contain the term, followed by taking the logarithm of the result. This combined TF-IDF score helps to down-weight the influence of frequent but less informative terms and up-weight the contribution of rare but potentially more meaningful terms. The final TF-IDF score is obtained by multiplying the TF value by the IDF value.

$$TFIDF_{(t,LT,D)} = TF_{(t,LT)} \times IDF_{(t,D)} \quad (1)$$

$$TF_{(t,LT)} = \frac{f_{(t,LT)}}{\sum f_{(t',LT)}} \quad (2)$$

$$IDF_{(t,D)} = \log\left(\frac{D}{|t \in D|}\right) \quad (3)$$

where:

- t: The token or word being evaluated,
- LT: The full text of the fetched license,
- D: The entire document corpus, in this case, the SPDX License List.

In this system, the TF-IDF agent is tasked with taking individual tokens (terms) from the fetched license text and performing the necessary calculations to generate a vector of TF-IDF values for each term. These vectors are then compared with the precomputed vectors from the SPDX license corpus using cosine similarity, as discussed in the previous section. The output is a ranked list of licenses based on the similarity score, with licenses exceeding a certain threshold returned as potential matches.

The use of TF-IDF allows for an efficient and accurate comparison of license texts by focusing on terms that hold the most informational value, thereby reducing the computational overhead associated with matching entire documents word-for-word. In this system, the TF-IDF calculation is implemented using the TfidfVectorizer module from the scikit-learn library (Pedregosa et al., 2011), which provides an efficient and scalable solution for handling large document sets and complex vector computations.

Algorithm 2 TF-IDF agent pseudocode

Require: $InputFile$
Ensure: $AllLicenses$

```
t ← loadFile(InputFile)
s ← TfidfVectorizer()
m ← s.fit_transform(AllLicenses)
v ← s.transform(InputText)
for l IN m do
    ss ← COSINESIMILARITY(l, v)
    if ss ≥ THRESHOLD then
        ADD license, ss TO sl
    end if
end for
return sl
```

The TF-IDF agent represents a balance between computational efficiency and retrieval precision.

By focusing on the most meaningful terms in the text, the algorithm is capable of accurately identifying relevant licenses even in cases where the exact wording of the license text may vary slightly across different files. This makes it particularly well-suited to the task of identifying open-source licenses, where boilerplate text may differ minimally but still contain critical legal distinctions.

3.4 DLD Agent

The Damerau-Levenshtein Distance (DLD) is an effective metric for calculating the similarity between two text sequences based on their edit distance. In this system, the DLD agent computes the Damerau-Levenshtein distance between the fetched license text and the original SPDX license list. The edit distance metric is particularly useful in scenarios where small textual variations—such as typographical errors or minor format changes—exist between two license texts.

Edit distance refers to the minimum number of operations required to transform one string into another. These operations typically include insertion, deletion, substitution, and transposition of characters. For example, the strings "License" and "Licensor" can be made identical by substituting the final 'e' in "License" with an 'o' and inserting an 'r' at the end, resulting in an edit distance of 2 (Kondrak, 2005). Originally proposed by Fred J. Damerau

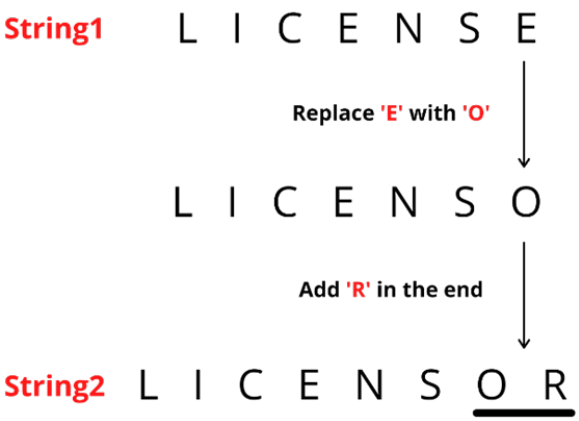

Figure 3: Illustration of Damerau-Levenshtein distance between two strings.

(Damerau, 1964), the Damerau-Levenshtein distance extends the traditional Levenshtein distance by accounting for common typographical errors. Damerau demonstrated that these errors can often be corrected by applying four fundamental operations: insertion, deletion, substitution, and transposition of two adjacent characters (Levenshtein, 1966). The DLD agent in this system leverages a Python library, "pyxDamerauLevenshtein," to compute the Damerau-Levenshtein distance. The agent compares the sequence of words in the fetched license text with those in the original SPDX license statements. However, despite its utility, this approach can be computationally expensive due to the size of the dataset. The DLD agent must check the token sequence of the fetched license against more than 400 SPDX licenses, with each license containing an average of 1,286 tokens. As a result, the complexity of this approach makes it one of the more resource-intensive algorithms in the system.

Algorithm 3 DLD agent pseudocode

Require: $InputFile$
Ensure: $AllLicenses$

```
t ← loadFile(InputFile)
d ← sys.maxsize
r ← 0
for l IN AllLicenses do
    x ← damerau_levenshtein_distance(t, l)
    if x < d then
        d ← x
        r ← license
    end if
end for
return r
```

The pseudocode above demonstrates the workflow of the DLD agent. The input file is compared against the original SPDX licenses by iterating through each license and computing the edit distance. The agent returns the license that has the smallest edit distance to the input text. This makes the DLD agent highly effective for scenarios where license texts are similar but exhibit slight variations due to transcription or formatting differences.

3.5 wordFrequencySimilarity Agent

The fact that there aren't too many variations of open-source license texts helps the wordFrequencySimilarity agent to give us the similarity score based upon the usage of unique words. In order for the text to belong to a certain license, it requires few conditions to meet, and this is the reason most of the statements are either left untouched or modified very little.

It takes into consideration each of the unique words present in the fetched license text and calculates its frequency. It then does the same with the original license texts and compares the similarity

Algorithm 4 wordFrequencySimilarity agent pseudocode

Require: $InputFile$, $AllLicenses$

```
function WORDFREQUENCY(d)
    return frequency counts of words in d
end function
function WORDFREQUENCYSIMILARITY(InputFile)
    i ← wordFrequency(load(InputFile))
    c, r ← 0, None
    for l IN AllLicenses do
        y ← histogram similarity between i and wordFrequency(l)
        if y > c then
            c, r ← y, l
        end if
    end for
    return r
end function
```

between both. The scores thus returned are based upon the similarity of the unique word frequencies. The system agent returns the results along with the similarity scores between the licenses in descending order. Although the algorithms seem too rudimentary but do an effective job of returning the top results along with the calculated similarity score.

3.6 N-Gram Agent

The N-Gram technique, commonly used in Natural Language Processing (NLP), involves a continuous sequence of terms or tokens within a corpus, where "N" denotes the number of terms grouped together. These tokens can include words, letters, phrases, or sentences, depending on the specific use case. N-Grams provide a substantial advantage over simple bag-of-words approaches by preserving the contextual ordering of terms, which is particularly valuable for license text analysis.

In a bag-of-words model, all words are stored without regard for their order, often resulting in a loss of context. By contrast, N-Gram models maintain this order, with the granularity determined by the value of N. For example, when $N = 1$, the result is a unigram (single word), while $N = 2$ forms a bigram, and so forth. This contextual retention allows N-Grams to accurately capture the semantics of language, which can be instrumental in identifying specific license texts that may differ only slightly in phrasing.

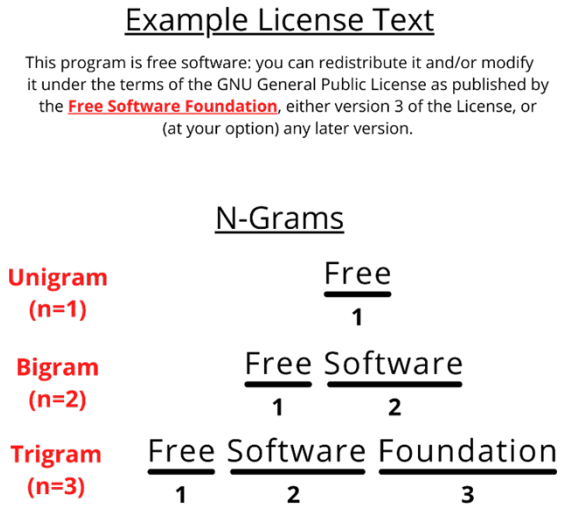

Figure 4: Examples of Unigram, Bigram, and Trigram for "Free Software Foundation"

To illustrate, consider the phrase "Free Software Foundation." For various values of N, the sequence breaks down as shown in Figure 4.

As another example, take the sentence "This software is licensed under MIT." Depending on N, the N-Grams would be as follows:

- For $N = 1$ (Unigrams): "This," "software," "is," "licensed," "under," "MIT."
- For $N = 2$ (Bigrams): "This software," "software is," "is licensed," "licensed under," "under MIT."
- For $N = 3$ (Trigrams): "This software is," "software is licensed," "is licensed under," "licensed under MIT."

Due to its simplicity, efficiency, and context-retaining abilities, the N-Gram model finds extensive applications in areas like NLP, computational linguistics, and computational biology. Its practical use cases span from spell-checking and text autocompletion to document summarization, making it well-suited for open-source license text analysis.

In this system, users have the option to employ cosine or dice similarity metrics for comparing the N-Grams of different license texts. During the initial build process, the N-Grams of all original license texts are generated and stored in a JSON file. This pre-processing step minimizes resource consumption during actual license detection, as the N-Grams of the input text are dynamically generated and then matched against those stored in the JSON file. This comparison yields a similarity score, allowing the system to efficiently identify the most relevant SPDX licenses associated with

the input.

This approach enables the detection tool to handle a vast array of license texts with minor variations by leveraging the inherent context-preserving properties of N-Grams, ensuring accurate and efficient license identification.

4 Evaluation of Scans

The evaluation process for our system involved a comprehensive test on over 1,400 files, derived from Nomos test data, an extensive collection used in open-source license detection research (FOSSology, 2021b). This dataset provided a robust benchmark to assess the accuracy and reliability of our system across varied license types.

For the purposes of this evaluation, `pom` and `xml` files were excluded due to their common tendency to only reference license names rather than contain complete license texts, which falls outside the scope of this scanner's capabilities. Files shorter than five lines were also removed to ensure that the analysis metrics reflect realistic and applicable scenarios. This filtering of test files yielded a curated dataset focused on license detection, particularly for complete license statements or headers.

Some of the entries within the Nomos Test Data repository utilize outdated license naming conventions, which prompted the removal of those entries that are not recognized within the current list of licenses processed by our tool. Furthermore, a normalization process was applied to standardize the license names, ensuring consistency across the dataset.

To systematically evaluate the performance of different algorithms, we utilized a modified evaluator script capable of calculating the precision metrics for each detection method. This script iteratively runs each algorithm on all files within the dataset, comparing each output with the known expected result to compute accuracy and `f1-score`. By automating this evaluation, we were able to produce a detailed accuracy metric for each detection method, providing insights into their respective strengths and weaknesses.

While effective, our system's reliance on automated detection methods introduces potential risks, particularly with nuanced or fragmented license texts. Misidentification of licenses may occur, emphasizing the importance of incorporating human oversight for critical compliance cases in future implementations.

Algorithm 5 Evaluator Pseudocode

Require: $agent$
Ensure: $TestData$

```
time.start()
m ← 0
for f in TestData do
    RUN(agent, TestData[f])
    if output = OriginalResult then
        m ← m + 1
    end if
end for
accuracy ← m/len(TestData)
return accuracy
```

The algorithm, as described in Algorithm 5, calculates the proportion of correctly identified licenses by comparing the output of each scan with the expected result. The metrics obtained provide a quantitative measure of each method's performance, contributing to a robust evaluation of the system's overall license detection accuracy.

5 Results

The license detection agents were evaluated using the Nomos test dataset, comprising over 1,400 labeled files (FOSSology, 2021b). This diverse dataset, containing various open-source license formats, served as a rigorous benchmark to assess each agent's accuracy. Key performance metrics, including precision, recall, and F1-score, were calculated to analyze each algorithm's effectiveness.

The software's command-line interface enabled recursive scanning, capturing all relevant files within nested directories. Future integration into a broader compliance framework with a GUI is planned to streamline usability and adoption. Ultimately, this tool aims to contribute to a comprehensive ecosystem for automated open-source license detection.

The tables below present accuracy and comparative performance metrics for each agent, showcasing the relative strengths of the different algorithms. These results provide empirical insights that will inform future optimization efforts. The individual metrics for each agent are as follows:

Table 1: Performance metrics of **Ngram** agent with **Cosine** similarity.

	Precision	Recall	F1-Score	Support
Accuracy	-	-	0.78	1116
Macro Avg	0.86	0.86	0.84	1116
Weighted Avg	0.88	0.78	0.78	1116

Table 2: Performance metrics of **Ngram** agent with **Dice** similarity.

	Precision	Recall	F1-Score	Support
Accuracy	-	-	0.86	992
Macro Avg	0.91	0.90	0.89	992
Weighted Avg	0.91	0.86	0.86	992

Table 3: Performance metrics of **Ngram** agent with **Bigram Cosine** similarity.

	Precision	Recall	F1-Score	Support
Accuracy	-	-	0.78	1070
Macro Avg	0.86	0.87	0.84	1070
Weighted Avg	0.86	0.78	0.79	1070

Table 4: Performance metrics of **DLD** agent.

	Precision	Recall	F1-Score	Support
Accuracy	-	-	0.26	1450
Macro Avg	0.47	0.31	0.34	1450
Weighted Avg	0.53	0.26	0.32	1450

Table 5: Performance metrics of **TF-IDF** agent with **Cosine** similarity.

	Precision	Recall	F1-Score	Support
Accuracy	-	-	0.83	1074
Macro Avg	0.91	0.91	0.89	1074
Weighted Avg	0.87	0.83	0.83	1074

Table 6: Performance metrics of **wordFrequencySimilarity** agent.

	Precision	Recall	F1-Score	Support
Accuracy	-	-	0.25	1448
Macro Avg	0.53	0.32	0.37	1448
Weighted Avg	0.57	0.25	0.30	1448

The evaluation results, as presented in Figure 5, indicate that the Ngram agent using Dice Similarity outperformed other agents, achieving an accuracy score of 86.2%, closely followed by the TF-IDF agent with Cosine Similarity, with an accuracy of 82.7%. The results highlight that agents based on cosine similarity generally yielded higher accuracy scores, averaging around 78-79%. In contrast, agents such as DLD, TF-IDF with Score Similarity, and wordFrequencySimilarity reported lower accuracy rates, indicating areas for potential improvement.

Overall, the evaluation demonstrates that the algorithms employed by the system exhibit a strong ability to accurately scan and identify open-source licenses, even when the variations between license texts are minimal. Further refinement of these agents is expected to enhance accuracy and establish the system as a reliable tool for open-source license compliance.

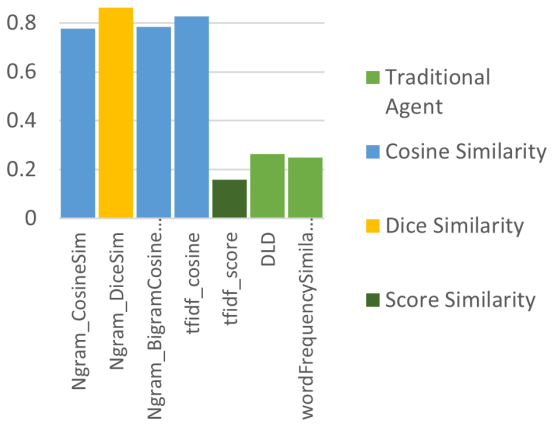

Figure 5: Accuracy scores of all the agents evaluated.

6 Future Plans

Our license detection tool, leveraging information retrieval and text similarity algorithms, has shown promising results. However, these methods may not fully capture subtle nuances among similar open-source license texts. To address this, future iterations will incorporate machine learning (ML) and deep learning (DL) algorithms, which can better handle minor textual variations and complex license structures. Effective ML and DL models require substantial data, so ongoing efforts within the open-source community are focused on curating a large and diverse dataset of license files (FOSSology, 2021a).

As we move towards ML integration, potential risks include increased computational demand and the possibility of overfitting, which could affect accuracy in unseen scenarios. Furthermore, plans include enhancing usability by integrating the tool into a broader compliance framework and optimizing performance through third-party libraries and code refinements.

The open-source community will play a key role in these advancements, providing essential feedback and support for continued accuracy and robustness improvements.

7 Limitations

This research, while advancing open-source license detection, is subject to several limitations. Firstly, the computational complexity of algorithms like Damerau-Levenshtein Distance presents challenges, particularly in terms of processing time and resource demands, which can limit scalability. Secondly, our evaluation utilized the Nomos test

dataset, which, while comprehensive, may not represent the full diversity of license texts encountered in real-world scenarios. Consequently, the generalizability of results to varied license forms remains uncertain.

The tool's reliance on text-based similarity measures may hinder performance with non-standard license formats or files containing significant non-license content, as these factors can affect text extraction and matching accuracy. Additionally, the current approach may struggle with distinguishing between licenses that have subtle variations or multiple licenses within a single file. Lastly, while future enhancements could incorporate machine learning for improved accuracy, such developments are constrained by the availability of labeled training data.

Addressing these limitations will be essential for refining the tool's efficacy, expanding dataset coverage, and enhancing the robustness of license identification in complex scenarios.

Acknowledgments

We extend our gratitude to the dedicated members of the open-source community who have contributed to this project. We also acknowledge the role of ongoing initiatives by organizations supporting open-source license compliance and standardization. Their support and collaboration have been instrumental in driving this research forward.

References

Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176.

FOSSology. 2021a. Minerva dataset generation.

FOSSology. 2021b. Nomos test case.

Free Software Foundation. 2019. REUSE specificaiton - 3.0.

Robert Gobeille. 2008. The FOSSology project. In *Proceedings of the 2008 international workshop on Mining software repositories - MSR '08*, page 47. ACM Press.

Michael C. Jaeger, Oliver Fendt, Robert Gobeille, Maximilian Huber, Johannes Najjar, Kate Stewart, Steffen Weber, and Adreas Würl. 2018. The FOSSology project: 10 years of license scanning. *International Free and Open Source Software Law Review*, 9(1):9–18.

Grzegorz Kondrak. 2005. N-gram similarity and distance. In Mariano Consens and Gonzalo Navarro, editors, *String Processing and Information Retrieval*, volume 3772, pages 115–126. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

J. Lerner. 2005. The scope of open source licensing. *Journal of Law, Economics, and Organization*, 21(1):20–56.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830.

G. Salton, A. Wong, and C. S. Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Hendrik Schoettle. 2019. Open source license compliance-why and how? *Computer*, 52(8):63–67.

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

Amit Singhal and others. 2001. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43.

Karenn Sparck Jones. 1972. A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL. *Journal of Documentation*, 28(1):11–21.

Christopher Vendome, Mario Linares-Vasquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. 2017. Machine learning-based detection of open source license exceptions. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 118–129. IEEE.

Nathan Wintersgill, Trevor Stalnaker, Laura A. Heymann, Oscar Chaparro, and Denys Poshyvanyk. 2024. "the law doesn't work like a computer": Exploring software licensing issues faced by legal practitioners. *Proc. ACM Softw. Eng.*, 1(FSE).

Hailing Zhang. 2020. Comparison of open source license scanning tools.