New pipeline for software mention extraction, disambiguation and enrichment (INRIA)(R,PU)

Deliverable information	
Deliverable number and name	D4.2
Due date	M18
Actual delivery date	M19
Work Package	WP4
Lead Partner for deliverable	Inria
Authors	Alain Monteil, Samuel Scalbert, Luca Foppiano, Martin Docekal, David Pride, Petr Knoth
Reviewers	Matteo Cancellieri (OU)
Approved by	
Dissemination level	Public
Version	

Document revision history			
Issue Date	Version	Author	Comments

Introduction

This document presents Deliverable 4.2 of the SoFAIR project, which focuses on the identification and evaluation of software mentioned in scientific publications. The work builds on resources developed by the CLARIN partner and extends previous efforts from the SoftCite project. In particular, this deliverable concentrates on applying and improving software mention recognition across two research domains that were central during the SoftCite project: Economics and Biology.

Currently, the software-mention data used for training was primarily sourced from these two fields. While this provided a strong foundation, it also introduced a disciplinary bias that limited the software's ability to generalize across other research domains. One of the goals of the SoFAIR project is to address this limitation by incorporating documents from a broader range of disciplines such as Digital Humanities (DH), Computer Science, Biomedical and Health Informatics (BMA), and others. This expansion aims to increase the diversity of language use, citation practices, and software usage patterns. By broadening the disciplinary coverage, SoFAIR seeks to create a more balanced and representative dataset, ultimately leading to more robust and accurate software mention detection across the research spectrum.

The primary objective of this deliverable is to assess how effectively software mentions can be detected in articles using machine learning-based approaches. To this end, we first tested a variety of DL and Large Language Model approaches using existing data. The project then extended the existing SoftCite software mention recognizer, integrating new annotated data that was generated by the CLARIN project partners. This enriched dataset served as the basis for retraining and evaluation, aiming to improve the performance of the current models.

The current software mention extraction pipeline includes four Deep Learning (DL) models, which are applied following GROBID-based preprocessing. Several enhancements were introduced to improve the robustness and domain adaptability of the system:

- Introduction of a new "**end-to-end**" **evaluation** using the SoMeSci[1] dataset to provide an unbiased evaluation;
- Introduction of a classification method for distinguishing between articles related and unrelated to software;
- Evaluation of alternative DL architectures and parameter configurations to identify the most effective model for software mention detection;

The outcome of this work is an updated extraction pipeline with retrained models capable of more accurately identifying software mentions in Economics and Biology publications, and increasingly in additional domains as disciplinary coverage expands. Evaluation was conducted using standard metrics, with a focus on achieving a sufficient F1-score in the target domains to justify deployment within the SoFAIR workflow in a semi-automated fashion.

[1] https://data.gesis.org/somesci/

This deliverable represents a key step toward enabling more accurate and scalable software metadata capture in scientific literature, contributing to the broader goals of the SoFAIR project.

Software mention extraction

Initial Model Experiments

In the early stages of the SoFAIR project, the annotation of a significant new multi-disciplinary corpus had not been completed. We therefore undertook a range of experiments with existing cross-disciplinary data. As the original SoftCite dataset is limited to two domains and the end goal of the project is to develop models that are as domain agnostic as possible, we wanted to conduct these early tests on data from multiple domains. For this study we employ a subset of the SoMeSci dataset for our experiments. The SoMeSci dataset consists of 399,942 triples representing 47,524 sentences from 1,367 documents. Overall the dataset contains high quality annotations (IRR: $\kappa = .82$) of 3,756 unique software mentions. A subset of 100 documents were selected[2] and the full text documents were collected from CORE.

Prior to passing textual information to the model for processing, the full text was extracted in its raw form and prepared for segmentation. Documents were split into individual sentences, paragraphs or supplied in their entirety. This step allowed us to test the effect of the level of text granularity given to the model. Three segmentation styles were tested: :

1. Sentence: Each sentence stands alone, without any overlap.
2. Paragraph: Groups of 20 sentences form each chunk, with the final two sentences in one chunk overlapping with the first two of the subsequent chunk.
3. Complete: The entire text is treated as one segment, with no internal splitting.

Once the text had been prepared, two individual pipelines were prepared for testing with various models. The simple approach processes the given text chunks without iterative refinement whereas the iterative approach uses both keyword-based and semantic filtering across multiple iterations to discover and verify new software mentions in the text.

Table 1 presents the results of our experiments with a range of different LLMs. Splitting the data and processing at the sentence level produced the best results overall, with paragraph-level splitting producing noticeably worse results across models.

[2] To speed up the processing.

Model	Pipe	Split type	Precision	Recall	F-Score
Llama3:70b	Simple	Sentence	0,732642	0,745163	**0,738849**
Gemma2:9b-instruct-fp16	Iterative	Sentence	0,746382	0,667775	**0,704894**
Gemma2:9b-instruct-fp16	Simple	Sentence	0,667067	0,694349	**0,680435**
Llama3:70b	Iterative	Sentence	0,67804	0,664426	**0,671164**
Sonnet	Iterative	Sentence	0,668872	0,670345	**0,669608**
Gemma2:27b-instruct-fp16	Iterative	Sentence	0,698903	0,637581	**0,666835**
Gemma2:9b-instruct-fp16	Simple	Paragraph	0,5775	0,686196	**0,627173**
Sonnet	Simple	Paragraph	0,567252	0,640409	**0,601615**
Llama3:8b	Iterative	Sentence	0,570989	0,603131	**0,58662**
Llama3.1:70b-instruct-q4	RAG	Sentence	0,589209	0,583002	**0,586089**
SciBERT - baseline	N/A	Paragraph	0,43299	0,414691	**0,423643**

Table 1: Top 10 performing models for software mention extraction

Our experiments with various models demonstrated that the best performing of these models (Llama3:70b) provides comparable performance to the best previous state of the art models in zero-shot set-up. This suggests that further improvements in performance can likely be attained with the addition of pre-training or fine-tuning of the LLM or by providing additional in-context examples (few-shot learning). Due to the generative nature of LLMs, we employed a position-invariant evaluation strategy, which differs from the traditional sequence labeling approach used in the Softcite experiments described in the following section. Specifically, we extract only the software mention itself, regardless of its position in the text, and consider it correct as long as it appears anywhere within the document.
In our experiments, it took in the range of 1-10 seconds (single-threaded) to process one document with some of our best performing models, the average being five seconds depending on document length. Assuming there are 10m new articles published every year, it would require in the range of 115 - 1,150 computational days to process all of them. This clearly demonstrates that, to enable adoption of these models in practice as part of open scholarly infrastructures, scalability is a critical issue. This would need to be achieved by either (a)

tolerating a slightly lower performance at the cost of higher speeds or (b) by having the financial resources to run these models in a distributed setup or (c) a combination of the above.

Improving the Software mention detection

The software mention detection application is built as a **cascade of models**, where each model builds on the output of the previous one to extract structured information from text. At a high level, the architecture begins with **sequence labelling models** that identify basic software-related spans in the text, followed by **specialised extractors** that assign deeper semantic roles or types to each identified mention.

Figure 1 illustrates this pipeline, where the first stage detects mentions of software and related metadata, and the second stage focuses on characterising these mentions by their functional type.

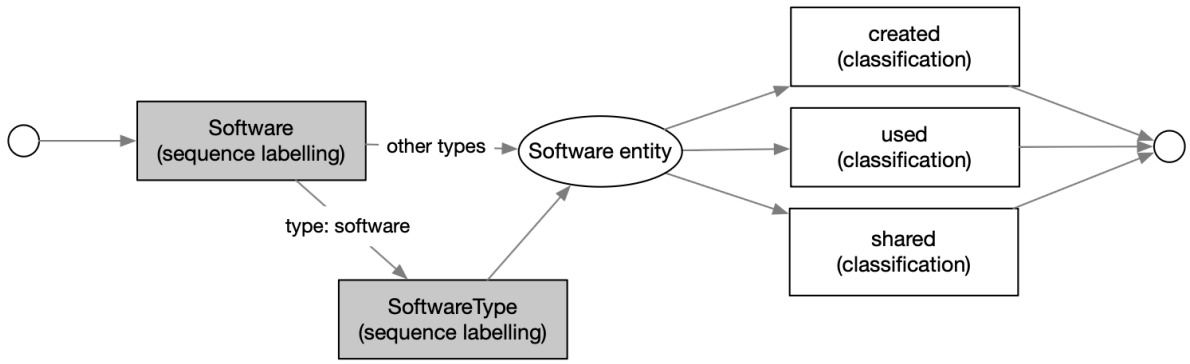

***Figure 1**: Software-mentions models cascade, with emphasis of the models software and softwareType that are the object of this document. In grey the models that are discussed in this deliverables. The figure shows how a software entity is built using the different models that are applied in cascade.*

Software-mentions workflow

The first sequence labelling model is designed to identify **software-related entities** (Figure 2) in text. These include:

- **Software** (primary information)
- **Version** of the software
- **URL** of the software, or the source code, for open source software, including Gitlab, Github, bitbucket
- **Creator**, which include persons, entities, companies, institutions

These components are identified through a **named entity recognition (NER)** approach, with the goal of finding both **seen**(known) and **unseen** (novel) software mentions in natural language text.

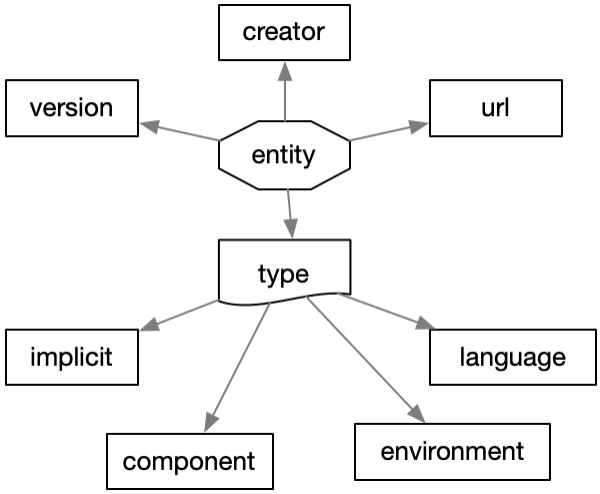

Figure 2*: Data model for the software entity*

Once a **software** mention has been extracted, the system performs **software type classification**, assigning a category that describes the **role or context** of the software. This includes:

- **Programming Language**: The mention refers to a mention intended as a programming language like Python or JavaScript.
- **Component**: The software is part of a larger environment or framework. For example, a script, a plugin or library used within an environment (see following)
- **Environment**: a software intended as a suite grouping several features, such as Python, R, Matlab, etc. Generally this is identified because the actual software is a script or a program that require the environment to run

In the example above, “Python” can be understood either as a programming language or as an environment, depending on the context in which it appears. For instance, the statement *“We used Python to develop our analysis”* refers to Python as a programming language. In contrast, *“We use the NumPy library in Python”* refers to Python as an environment, one that includes the programming language but also encompasses its broader ecosystem of libraries and tools.

An important detail is that some software mentions are **implicit**, that is the software is not directly named but inferred through context. For instance, in the sentence: *"Statistical analysis was performed with a script in R.",* the token “*script”* may implicitly refer to a program without naming them individually. While implicit mentions are often ignored, they can contribute positively in evaluating open science metrics, for example establishing whether a certain study has reused other unnamed software.

The system uses contextual cues and learned patterns to handle these cases, although they are more challenging than explicit mentions.

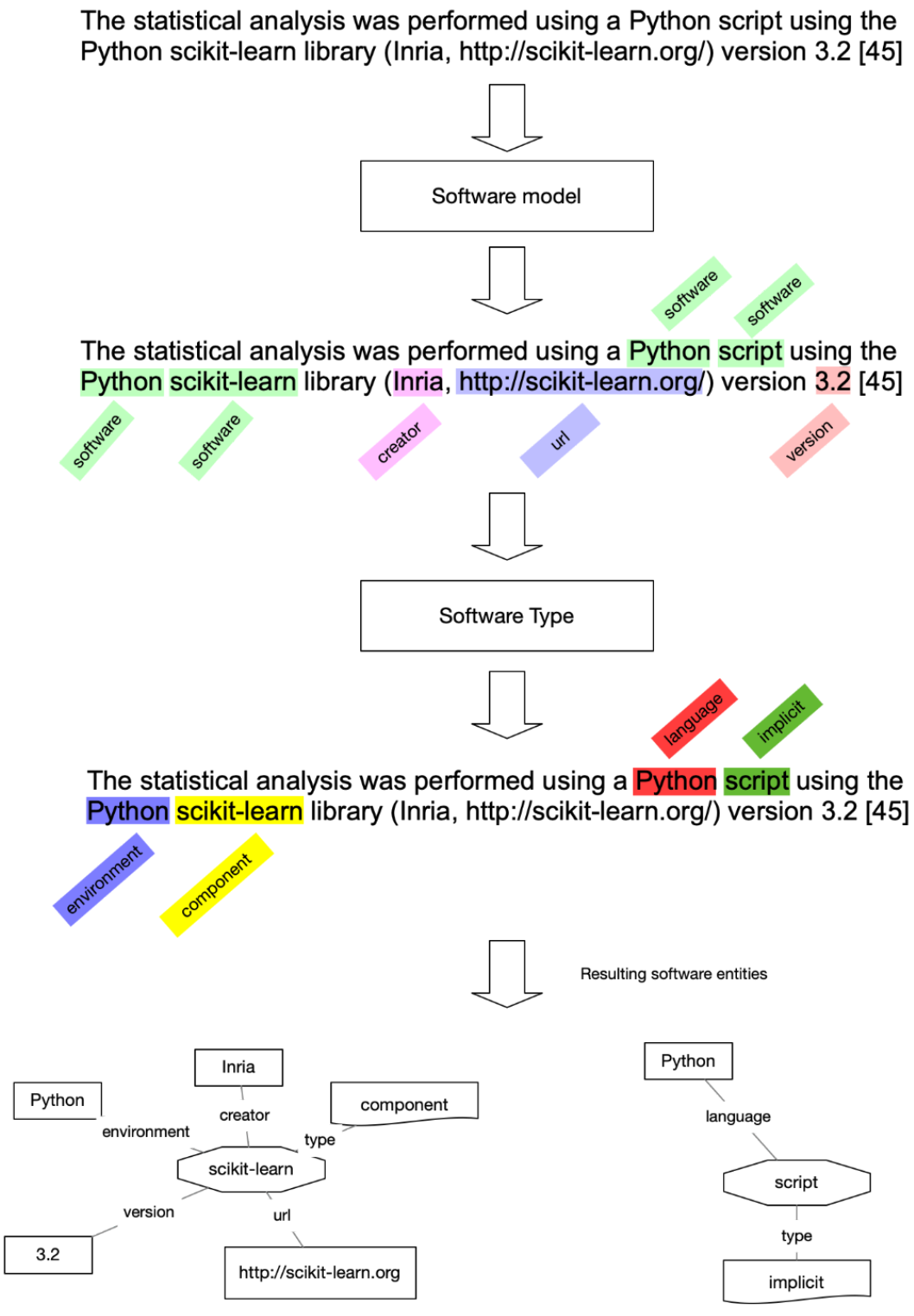

Figure 3a: Software mentions extraction (sequence labelling) applied on a real case example[3]

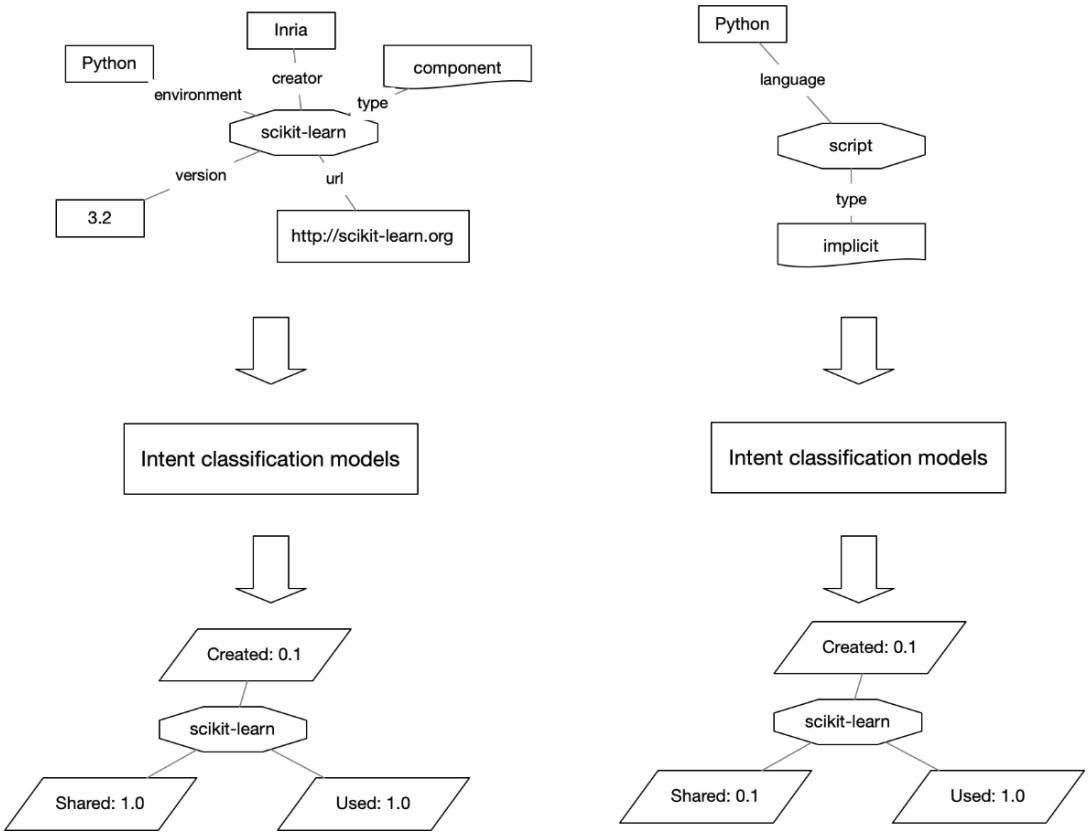

Figure 3b: Software mentions characterization (intent classification) applied on a real case example

Limitations

The model accuracy achieved in the SoFAIR project was influenced by the quantity and breadth of the available annotations. While considerable effort went into producing the manual annotations, during the course of the evaluation it was realised that the overall distribution could have adhered more closely to the article discipline distribution, thus balancing the selection of the SoFAIR dataset which may have provided a more focused dataset. As these systems rely heavily on data-driven models, their ability to improve and generalize is inherently tied to the richness and diversity of the annotated datasets they are trained on. Additionally, an unanticipated round of verification was required to ensure annotation quality, which was not originally foreseen in the project planning. As with many data-driven systems, performance is closely tied to the size, diversity, and quality of the training data. The results presented below

[3] The creator of scikit-learn is, in reality, not limited to the Inria institute. This simplified approach is used for illustration purposes only.

indicate that future improvements can be realised by expanding and refining the annotated dataset to support broader generalization and stronger model performance.

Method

Data collection

The original SoftCite dataset consists of a total of 4,971 documents, which are divided into two main subsets for machine learning purposes. The training set comprises 3,976 documents and is used to develop and fine-tune models. The holdout set contains the remaining 995 documents and is reserved for evaluating model performance on unseen data. This division supports rigorous experimentation and ensures that models generalize well beyond the training data.

The dataset was reduced due to the need for further validation of the initial annotations provided by CLARIN. These annotations were created without a feedback loop and, upon review, were found to contain a substantial number of discrepancies when compared with the original Softcite training data. Including them without correction could have introduced inconsistencies in the training process, potentially affecting model performance.

SoFAIR dataset

The SoFAIR dataset comprises a total of 261 TEI-validated and deduplicated documents which contains 3922 total annotations, structured to support robust model training and evaluation. Of these, 214 documents (3191 annotations) are allocated to the training set, providing the primary data for model development. The remaining 47 documents (731 annotations) form the holdout set, used to assess model performance on unseen data. The split is based on field-level calculations, where fractional values are rounded down using the floor function, ensuring that even minimal counts are consistently treated as the lowest whole number. This careful partitioning maintains dataset integrity while enabling reproducible and fair evaluation. The total size of the complete SoFAIR dataset reflects a deliberate trade-off between annotation quality and project resources. The annotation process was both time-intensive and domain-specific, requiring expert input to ensure consistency.. In practice, the project team prioritised producing a high-quality, fully TEI-compliant dataset, at the cost of a smaller overall size. The project team also undertook an added verification phase to enhance reliability. As such, the dataset

represents a valuable and reliable resource for model training.

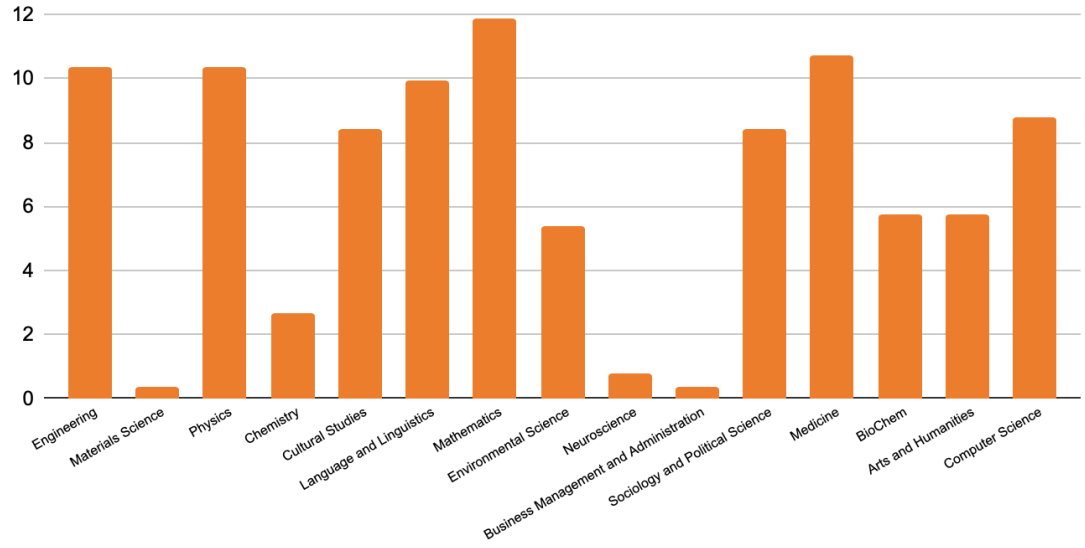

***Figure 4**: Distribution (in percentage) of papers among the different domains in the SoFAIR annotated corpus*

Field	Paper Count	20% (Test)	80% (Train)
Engineering	27	5	22
Materials Science	1	0	1
Physics	27	5	22
Chemistry	7	1	6
Cultural Studies	22	4	18
Language and Linguistics	26	5	21
Mathematics	31	6	25
Environmental Science	14	2	12
Neuroscience	2	0	2
Business Management and Administration	1	0	1
Sociology and Political Science	22	4	18
Medicine	28	5	23
BioChem	15	3	12
Arts and Humanities	15	3	12
Comp Sci	23	4	19
Total	**261**	**47**	**214**

Table 2: distribution per fields for training and test data

Results

In the following section we present and discuss the results obtained by the experiments of training and evaluation of the machine learning models using different architecture and combination of data for training and evaluation.

Sequence labelling evaluation

As discussed before, the machine learning models experimented with, are the “software” and “software type”.
To provide a better grasp of the respective size of each dataset or split, we provide as reference the number of occurrences of the most frequent label in the data.

We designed four different experiments:

- **Baseline**: we trained and evaluated using the SoftCite dataset: the train (most frequent annotation contains 4181 occurrences) and evaluation splits (989 occurrences), results are in Table 3 for the “software” model and Table 6 for the “software type” model.
- **SoftCite+All_SoFAIR**: we trained using the SoftCite dataset train split and the full SoFAIR dataset (in total, the most frequent annotation contains 6,642 occurrences) and evaluated using the SoftCite evaluation split (989 occurrences), results are in Table 4 for the “software” model and Table 8 for the “software type” model.
- **Combined**: we trained using the SoftCitetrain split and the SoFAIR train split (in total, the most frequent annotation contains 7,239 occurrences) and evaluated using the SoftCite evaluation split (989 occurrences) and the SoFAIR evaluation split (599 occurrences), results are in Table 5 for the “software” model and Table 7 for the “software type” model.
- **SoFAIR**: we trained and evaluated using the SoFAIR dataset: the train (most frequent annotation contains 2541 occurrences) and evaluation splits (599 occurrences), results are in Table 6 for the “software” model and Table 10 for the “software type” model.

The evaluation reveals important trends when comparing the different experimental setups, with the average F1-score serving as the primary metric for performance comparison.

Across the different configurations for the "software" entity recognition task (Tables 3,4,5 and 6), the average F1-scores are relatively consistent. The highest F1-score (0.7542) is observed in the SoftCite+All_SoFAIR model. This indicates that supplementing the SoftCite training data with the entire SoFAIR dataset offers a modest but measurable improvement in model performance. The improvement is especially relevant given the increased diversity in training examples, which likely enhanced the model’s generalization on the SoftCite test set.However, the Combined model, which is using slightly less training data than SoftCite+All_SoFAIR, does not yield higher or similar performance (F1-score: 0.7289). This suggests that the SoFAIR dataset may introduce inconsistencies or noise, offsetting the potential benefits of added data. This result emphasizes that dataset compatibility and label consistency are crucial when merging corpora.

The results for the "software type" classification task show greater variation in average F1-scores across the different experiments. As with the "software" model, the SoftCite+All_SoFAIR model configuration again yields the highest average F1-score (0.7739), although only marginally higher than the SoftCite-only baseline (0.7726). This demonstrates that the additional examples from SoFAIR contributed slightly to performance, likely by providing

more varied instances for rarer labels such as *component* and *implicit*. In contrast, the Combined model shows a significant drop in performance (F1-score: 0.6310), suggesting that evaluating on a merged test set introduces complexity that the model trained on combined data cannot handle effectively. The most likely explanation is inconsistency between the annotation styles or entity distributions across the two datasets, which could have led to confusion during both training and evaluation. The SoFAIR-only model, with an average F1-score of just 0.3955 strongly indicates that the SoFAIR dataset alone lacks sufficient volume or clarity to support effective learning for software type classification. Particularly, the lower support and higher ambiguity of certain labels (e.g., *implicit*) likely contributed to the poor performance.

Entity	Precision	Recall	F1-Score	Support
creator	0.7932	0.7520	0.7721	250
software	0.8021	0.6067	0.6908	989
url	0.5116	0.5366	0.5238	41
version	0.8783	0.8163	0.8462	283
All (micro avg.)	**0.8064**	**0.6660**	**0.7295**	**1563**

Table 3: Evaluation scores of the **baseline** “software” model, trained and evaluated with the SoftCite dataset’s train and test splits

	Precision	Recall	F1-score	Support
creator	0.8066	0.784	0.7951	250
software	0.7871	0.6542	0.7145	989
url	0.65	0.6341	0.642	41
version	0.8791	0.8481	0.8633	283
all (micro avg.)	**0.8048**	**0.7095**	**0.7542**	**1563**

Table 4: Evaluation scores of the **SoftCite+All_SoFAIR “**software” model**,** trained with SoftCite train split and all SoFAIR dataset, and evaluated on SoftCite test split

Label	Precision	Recall	F1-Score	Support
creator	0.7861	0.6193	0.6928	1584
software	0.7818	0.7517	0.7665	286

`url`	0.8719	0.8575	0.8646	365
`version`	0.5577	0.5686	0.5631	51
All (micro avg.)	**0.7952**	**0.6728**	**0.7289**	**2286**

Table 5: Evaluation scores of the **Combined** “software” model trained with SoftCite and SoFAIR dataset (train split) and evaluated with SoftCite and SoFAIR dataset (test split)

Label	Precision	Recall	F1-Score	Support
`creator`	0.4839	0.4167	0.4478	36
`software`	0.8043	0.4353	0.5649	595
`url`	0.3571	0.5	0.4167	10
`version`	0.8649	0.7805	0.8205	82
All (micro avg.)	**0.7778**	**0.4744**	**0.5893**	**723**

Table 6: Evaluation scores of the **SoFAIR** “software” model, trained and evaluated with the SoFAIR dataset’s train and test splits

Label	Precision	Recall	F1-Score	Support
`environment`	0.7947	0.8207	0.8075	184
`component`	0.6	0.6	0.6	15
`language`	0.625	0.6667	0.6452	15
`implicit`	0.5	0.625	0.5556	8
All (micro avg.)	**0.7576**	**0.7883**	**0.7726**	**222**

Table 7: Evaluation scores of the **baseline** “software type” model trained and evaluated with the SoftCite dataset

Label	Precision	Recall	F1-Score	Support
`environment`	0.8436	0.8207	0.832	184
`component`	0.4167	0.6667	0.5128	15
`language`	0.5714	0.5	0.5333	8
`implicit`	0.4643	0.8667	0.6047	15

All (micro avg.)	**0.7479**	**0.8018**	**0.7739**	**222**

Table 8: Evaluation scores of the **SoftCite+All_SoFAIR "**software type" model trained with SoftCite train split and all SoFAIR dataset, and evaluated on SoftCite test split

Label	Precision	Recall	F1-Score	Support
`environment`	0.7743	0.7543	0.7642	232
`component`	0.5333	0.4267	0.4741	75
`language`	0.5357	0.5769	0.5556	26
`implicit`	0.5882	0.2326	0.3333	86
All (micro avg.)	**0.6954**	**0.5776**	**0.631**	**419**

Table 9: Evaluation scores of the **Combined** "software type" model trained with SoftCite and SoFAIR dataset (train split) and evaluated with SoftCite and SoFAIR dataset (test split)

Label	Precision	Recall	F1-Score	Support
`environment`	0.4314	0.4583	0.4444	48
`component`	0.4364	0.4	0.4174	60
`language`	0.6429	0.5	0.5625	18
`implicit`	0.381	0.2254	0.2832	71
All (micro avg.)	**0.4383**	**0.3604**	**0.3955**	**197**

Table 10: Evaluation scores of the **SoFAIR** "software type" model trained using the SoFAIR dataset, using the train/test splits described above

End to end evaluation

In this section we provide the end to end evaluation using the SoMeSci (https://arxiv.org/pdf/2108.09070) dataset to assess, in a non-biased way, how the overall application generalises on data outside its domain of training and evaluation.
Moreover, the end to end evaluation takes in account the full pipeline, and provides a single metric that also comprehends possible software bugs and propagated errors during the processing.
Since the SoMeSci dataset was created by an independent team, we adapted the evaluation to compare only comparable entities, resulting in the four main labels: creator, software, url, version.

The results in Table 12 indicate an improvement in the model's generalisation ability when trained on both the SoftCite and SoFAIR datasets, as compared to the baseline model trained solely on SoftCite (Table 11).
This is most evident in the overall F1-score, which increased from 0.6317 to 0.6796 (~+9%), demonstrating the model's enhanced capacity to perform well on unseen data from the SoMeSci corpus.
Individual label performance also reflects this trend. For example, the 'creator' and 'version' labels showed around 4% F1-score gains (from 0.7263 to 0.7668 and 0.8122 to 0.8548, respectively), indicating that exposure to a more diverse training set enabled the model to better capture varied patterns. The 'software' class also showed a slight drop in precision, which suggests that the model began identifying more instances as software, with a higher chance of including incorrect predictions, nevertheless, showing overall improvements in both recall and F1-scores.
The overall metrics point to an enhancement in the model's ability to generalise across different scientific corpora, reinforcing the importance of incorporating heterogeneous training data for robust citation extraction.

Label	Precision	Recall	F1-Score	Support
`creator`	0.8734	0.6216	0.7263	111
`software`	0.7037	0.4940	0.5806	423
`url`	0.1667	0.08	0.1081	25
`version`	0.93	0.7209	0.8122	128
All (micro avg.)	**0.7566**	**0.5422**	**0.6317**	**688**

Table 11: Baseline models trained on SoftCite and evaluated on the SoMeSci

Label	Precision	Recall	F1-Score	Support
`creator`	0.9024	0.6667	0.7668	111
`software`	0.6745	0.6123	0.6419	423
`url`	0.1111	0.08	0.0930	25
`version`	0.9196	0.7984	0.8548	129
All (micro avg.)	**0.7288**	**0.6366**	**0.6796**	**688**

Table 12: Model trained on SoftCite and SoFAIR and evaluated on the SoMeSci

Processing throughput evaluation

In this section we report the measured throughput of the service in terms of document per seconds, using the three different sources of input the data: PDF document, XML-TEI format, or plain text (TXT). The experiment was performed by processing 100 documents on a machine whose parameters are illustrated in Table 9.

The experiments were performed using several configurations:

- **Concurrency**: the number of documents sent in parallel by the client, we tested with 8, 16, and 24 concurrent documents for the single service, and with 8, 16, 24, and 32 concurrent documents for the 2x service
- **Architecture**: the Deep Learning architecture used for the software model (the most heavily used model in the chain):
    - **BERT**: is the standard BERT (Bidirectional EncodeR for Text) with a standard activation layer
    - **BERT_CRF** is the same architecture with a CRF activation layer, which offers better results at the price of a smaller throughput

The experiment was performed using a single service and a cluster of two services coordinated with a load balancer.

CPU model	Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz
CPU sockets	2
CPU cores	24
CPU threads	48
CPU max MHz	3200.0000
RAM [GiB]	187.6
GPUs	2x NVIDIA GeForce RTX 2080 Ti
GPU memory [MiB]	2 x 11264 Mb
OS	Ubuntu 20.04.6 LTS

Table 13: Overview of the service on which SoftCite was deployed

Table 13 presents the benchmarking results for processing 100 documents across various configurations. Among the three formats, PDF processing is consistently the slowest, primarily due to the additional overhead of PDF transformation and structuring.The TXT processing also shows slower performance, while XML-TEI achieves the fastest processing times. These results

align with expectations. The processing of TXT documents was originally designed for testing purposes only; however, in our experiments, we processed the full text of entire articles. These texts likely exceed the 512-token limit of BERT-based models and therefore must be split into individual sentences to avoid information loss, which significantly increases the number of sequences that need to be processed. In contrast, the XML-TEI format already provides structured content, with paragraphs presented as discrete text segments. As a result, even though some long paragraphs are split into sentences, the total number of input sequences remains lower than in the TXT format for equivalent documents. This enables longer average input segments and leads to faster overall processing for XML-TEI.

From a throughput perspective, XML-TEI is consistently the most efficient format for large-scale processing, achieving up to 0.6 documents/second on a single node and 1.14 documents/second on a dual-node configuration. This demonstrates the advantages of using a pre-processed XML-TEI pipeline, where structural parsing (e.g., via Grobid) is performed in advance, enabling downstream extractors like software-mention taggers to run more efficiently. Nonetheless, PDF remains the reference format when enriched outputs are required, for example, for directly annotating software mentions and their characterizations within the original PDF document.

In terms of architecture, our results indicate that BERT consistently outperforms BERT+CRF in processing speed across all formats and configurations. The addition of a Conditional Random Field (CRF) layer introduces overhead due to the increased computational complexity in sequence decoding. BERT+CRF may offer marginal gains [5] in tagging accuracy for some tasks due to the Conditional Random Field layer as activation layer. Nevertheless, in our tasks, the results are comparable (F1 of 73.06 using BERT vs 72.95 using BERT_CRF)
However, there is a general and significant performance trade-off with throughput boost ranging from 20% to over 40% depending on the input format and concurrency level. Therefore, for high-throughput or large-scale processing scenarios, vanilla BERT offers a more efficient and scalable solution.

Nodes	**Conc**	**Architec**	**PDF**		**XML-TEI**		**TXT**	
			doc/s	s	doc/s	s	doc/s	s
Single		BERT	0.28	352	**0.59**	169	**0.38**	261
	8	BERT+CRF	0.19	540	0.30	334	0.20	494
		BERT	**0.30**	338	0.58	171	0.37	267
	16	BERT+CRF	0.17	598	0.30	338	0.20	490
		BERT	0.29	342	0.63	160	0.36	277
	24	BERT+CRF	0.16	644	0.30	336	0.20	500
Multi (2)		BERT	0.53	187	1.11	90	0.66	152
	8	BERT+CRF	0.41	243	0.55	182	0.37	270

		BERT	**0.54**	185	**1.14**	88	**0.67**	150
	16	BERT+CRF	0.39	256	0.54	185	0.38	266
		BERT	0.43	235	0.98	102	0.64	156
	24	BERT+CRF	0.33	306	0.55	182	0.32	314

Table 14: Throughput results measured in document per seconds with several conditions: concurrency, architecture and number of nodes. The reported values represent the average of 3 runs.

Software-related Documents classification

To optimize our software mentions extraction pipeline, we analyzed data from **SoFAIR**, **SoftCite**, and **SoMeSci**. We found that only **38%** of documents contained at least one software annotation. This indicates that a significant portion of documents are being processed unnecessarily, despite the computational cost, our pipeline is able to process only a fraction of a document per second using single node setup.

To address this inefficiency, we developed a filtering model that identifies documents likely to contain software mentions, enabling more targeted and scalable processing of large document collections.

We trained a classification model, **ModernBERT-base**[4], on the combined SoFAIR, SoftCite, and SoMeSci datasets. ModernBERT was chosen because it is a modern implementation of the BERT architecture providing several improvements, including throughput (by supporting flash attention 2) and context length (with rotary positional embeddings - RoPE). Unfortunately, the same architecture could not be used for the other models, due to incompatibility with the libraries used for the current integration into the software-mention infrastructure.

The model distinguishes between documents with and without software mentions. Evaluation results are shown in the table below. In our benchmark, the model yielded a throughput of **39 documents per second** on a single **NVIDIA GeForce RTX 4090** GPU.

Meaning that for a setup with one filter instance (*f=39 doc/s*) and two Softcite instances processing TEIs (*c=1.14 doc/s*), the whole pipeline with filter selecting *p=38%* of documents will be able to process *2.79 doc/s*, which is a *145%* speed up.

precision	0.8625
recall	0.9104
f1	0.8858
accuracy	0.9268

Table 15: ModernBERT-base results on a test set from the collection of SoFAIR, SoftCite, and SoMeSci data.

The model is publicly available at https://huggingface.co/SoFairOA/sofair-modernBERT-base-filter, and we also provide a

[4] https://huggingface.co/blog/modernbert

lightweight command-line tool (https://github.com/SoFairOA/filter) for easy integration into existing workflows.

Software disambiguation

Introduction

Disambiguation plays a crucial role in accurately interpreting software mentions within scientific articles. In the Softcite dataset, the data were analyzed not in isolation but collectively within the same article. This approach allows for a more nuanced understanding of the author's intent, whether the software is being used, created, or shared, by considering the broader context of all references throughout the text. Grouping mentions at the article level and providing a *normalized form* (a plain text form) enhances the accuracy of intent classification, as individual sentences may only provide partial or ambiguous signals [2].

Grouping mentions across documents enhances the accuracy of intent classification, as isolated mentions within a single sentence or article often provide only partial or ambiguous signals. By aggregating software mentions across articles, we can uncover broader trends that are usually hidden when analyzing documents in isolation. However, the wide variation in how software is referenced is shaped by discipline, writing style, and author preferences which presents a significant challenge. These diverse expressions require sophisticated disambiguation strategies to ensure consistent identification and interpretation of software mentions across a heterogeneous corpus.

Approach to disambiguation

A software's normalized form refers to a standardized version of its name. This normalization ensures consistency by removing variations such as abbreviations, case differences, or misspellings, and mapping these variations to a canonical form. The goal is to support accurate identification and citation of software across documents.

Despite implementing normalization, we observed that inconsistencies in software naming still persisted across our corpus. To address this, we introduced a multi-step disambiguation process, beginning with fuzzy matching.

Fuzzy matching

The first level of disambiguation uses the Python library Fuzzy, which relies on Levenshtein distance, a metric that measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another.

For example, strings like *"Python"*, *"python"*, and *"Py-thon"* would each receive a similarity score above 90 (out of 100), indicating a high degree of similarity. In our corpus, we apply fuzzy matching against each software's normalized form to identify similar variants. Mentions with a similarity score above 90 are flagged for verification.

However, even with this automated step, the number of software mentions requiring manual review quickly became unmanageable, even within our relatively small corpus. Additionally, fully automating the verification and correction process proved unreliable, as software names are inherently dynamic: they may evolve over time, change in format, or vary by version and context.

Author-Guided Verification

To increase the accuracy of the automatic pipeline, we incorporated an author-in-the-loop approach. After each deposit, a pipeline processes the associated document, prepare an automatic aggregation, and sends a notification to the submitting author, inviting them to review the extracted software mentions. This human-in-the-loop step ensures we capture the correct canonical form of each software and, where possible, the corresponding repository or URL.

With this verified data, we can enrich our database by linking software to research teams, authors, or relevant external resources. Over time, as the database grows, it will improve our ability to disambiguate software automatically, reducing the need for manual intervention. It also enables us to pre-fill verification requests, further easing the burden on authors.

This process creates a virtuous cycle: as the database becomes more robust, it supports better disambiguation, leading to higher-quality data, improved software visibility, and a more accessible, findable scholarly record.

Towards semantic embeddings and contrastive learning

Looking forward, high-quality, author-verified annotations open the door to more sophisticated methods of disambiguation such as semantic comparison and contrastive learning. Rather than relying solely on surface-level string similarity, these approaches learn deeper representations of software mentions in context,capturing the semantics of how software is described and used. However, these models require a reliable ground truth to function effectively. A small but accurate, curated corpus that is built with the author-in-the-loop process is essential as a foundation for training and validating such models. Only with this solid base of trustworthy annotations can we confidently scale to semantic-level disambiguation, enabling models to distinguish between truly distinct software entities even when they share similar or ambiguous names.

Data availability

All the data and code including all the trained models, the created and reused datasets, and the scripts for evaluations, are available on huggingface at https://huggingface.co/SoFAIROA. The software-mentions code is available on GitHub at https://github.com/softcite/software-mentions.

Conclusions and perspectives

This deliverable presents significant advancements in the detection, disambiguation, and enrichment of software mentions in scientific literature, forming a cornerstone of the SoFAIR project's mission to enhance software transparency and traceability in open science.

Zero-shot
We did experiments with various LLMs in a zero-shot setup. Even though we got promising results (up to .74 F1), we identified that scalability is an issue for open scholarly infrastructures. We thus focused on smaller models using supervised fine-tuning.

SoFAIR impact in software mention extraction:
The Software-mention application used for mining the SoftCite dataset, was considered a solid base for improving the software mention extraction especially considering disciplines outside the initial scope of SoftCite, limited to economics and biology. SoFAIR aims to cover disciplines included, but not limited to, materials science, digital humanities, chemistry, business and administration.
Evaluation across multiple experimental settings demonstrated that supplementing the SoftCite dataset with SoFAIR data (SoftCite+All_SoFAIR) yields the best performance, with F1-scores reaching **0.7542** for software recognition and **0.7739** for software type classification. However, inconsistencies in annotation across datasets negatively affected the performance of models trained on combined corpora. The SoFAIR-only model showed limited effectiveness due to smaller training size and higher ambiguity, especially in the software type task.

Nevertheless, the results of the end to end evaluation using the independent SoMeSci dataset, showed that the combined model maintained strong generalisation performances. Despite the SoFAIR dataset being significantly smaller than the SoftCite dataset, its inclusion during training led to noticeable improvements in model performance and generalisation (Tables 11 and 12). The overall micro-averaged F1-score improved from **0.6317** to **0.6796** and highlights a reliable level of performance on the diverse and unseen SoMeSci dataset. Notably, key categories such as 'creator' and 'version' saw substantial gains in F1-score, while 'software' improved in recall at the expense of a slight drop in precision, indicating the model was able to identify more relevant instances, albeit with some trade-offs. These results suggest that even limited but complementary and diverse training data like SoFAIR can meaningfully enhance a model's ability to generalise across domains.

Throughput and Scalability:
Processing performance evaluations revealed XML-TEI as the most efficient input format,

enabling faster throughput than PDF or plain text. The system demonstrated solid scalability under different concurrency levels and hardware configurations, with a processing rate of up to **1.14 documents/sec** for XML-TEI using multi-node deployment and the standard BERT architecture. The BERT architecture resulted to be constantly 20-40% faster than the slightly more accurate BERT_CRF architecture.

Document Classification for Pre-filtering:
To increase system efficiency, we developed a classification model (ModernBERT-base) to pre-filter documents likely to contain software mentions. This model achieved an **F1-score of 0.8859** and processed **39 documents/second**, substantially reducing computational overhead, and enabling scalable processing for large archives. We estimated that the filter provides **145%** speed up for the whole pipeline.

Software Disambiguation and Enrichment:
A multi-tier disambiguation pipeline was implemented, beginning with fuzzy matching and extending to an **author-in-the-loop** verification system. This process enables high-precision enrichment by linking software mentions to canonical names, URLs, and repositories. Verified data also lays the groundwork for future semantic disambiguation via contrastive learning methods. Author feedback is central to refining the dataset and improving disambiguation accuracy over time.

Integration and Outlook:
Work is underway to integrate the software mention pipeline into open infrastructures such as HAL and CORE, using COAR Notify to engage authors in the validation loop. This creates a sustainable ecosystem for capturing, verifying, and reusing software metadata across research domains. As the author-verified database grows, it will support more accurate, scalable, and semantically rich disambiguation and classification models. All the detailed integration, workflows and pipelines with other partners of the project including CORE and Inria are described in detail in the general documentation, deliverable 3.2 https://sofairoa.github.io/documentation/.

In summary, the work conducted in this deliverable strengthens the SoFAIR infrastructure for software metadata extraction and lays the foundation for future enhancements in machine learning-based disambiguation, human-in-the-loop validation, and integration with research data infrastructures. The tools, data, and models produced are openly available and contribute directly to promoting FAIR software practices in scholarly communication.

References

1. Schindler D, Bensmann F, Dietze S, Krüger F. Somesci-a 5 star open data gold standard knowledge graph of software mentions in scientific articles. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management; 2021. p. 4574-83.

2. Lopez P, Du C, Cohoon J, Ram K, Howison J. Mining software entities in scientific literature: document-level ner for an extremely imbalance and large-scale task. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management; 2021. p. 3986-95.

3. Du C, Cohoon J, Lopez P, Howison J. Softcite dataset: A dataset of software mentions in biomedical and economic research publications. Journal of the Association for Information Science and Technology. 2021;72(7):870-84.

4. Istrate AM, Fisher J, Yang X, Moraw K, Li K, Li D, et al. Scientific Software Citation Intent Classification Using Large Language Models. In: International Workshop on Natural Scientific Language Processing and Research Knowledge Graphs. Springer Nature Switzerland Cham; 2024. p. 80-99.

5. Foppiano, L., Castro, P. B., Ortiz Suarez, P., Terashima, K., Takano, Y., & Ishii, M. (2023). Automatic extraction of materials and properties from superconductors scientific literature. Science and Technology of Advanced Materials: Methods, 3(1). https://doi.org/10.1080/27660400.2022.2153633