



Department of Management Science & Technology

MSc in Business Analytics

**“Towards streamlining Reproducibility
Studies in Academic Research”**

By

Evrydiki Vrouvaki

Student ID Number: f2822302

Name of Supervisor: Charis Papageorgiou

March 2025

Athens, Greece



ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis supervisor Prof. Haris Papageorgiou for his guidance, expertise, and support throughout the research process. Special thanks to Petros Stavropoulos for providing constructive feedback, thus guiding me towards academic excellence. He provided me with assistance for troubleshooting complex issues to ensure that the process ran smoothly.

Last but not least, I want to thank my family and friends for their continuous encouragement, support and understanding during this challenging period.





ABSTRACT

The focus of this master thesis is to address the difficulty of combining and standardizing annotations across diverse Natural Language Processing tools, particularly when analyzing scientific literature. The suggested solution focuses on a generalized and mapping-based approach. This approach maps the research artifacts in sentence level from the Research Artifact Analysis tool to CAS files that are created within the INCEpTION platform. This integration is crucial in order to deal with inconsistencies in sentence segmentation, tokenization, and encoding formats, which are important barriers to reliable annotation mapping. In order to find the most accurate sentence alignments, this study makes use of text normalization techniques and the Longest Common Subsequence algorithm. In order to guarantee high annotation accuracy, the framework also includes mechanisms that use case-insensitive matching and alternative methods. Annotations for research artifacts and their associated metadata are programmatically added into the CAS structure. This research contributes to the scientific community by providing an adaptable framework in order to improve annotation consistency across NLP tools. Its outcomes support the development of more reliable and automated workflows, with potential applications in many fields.



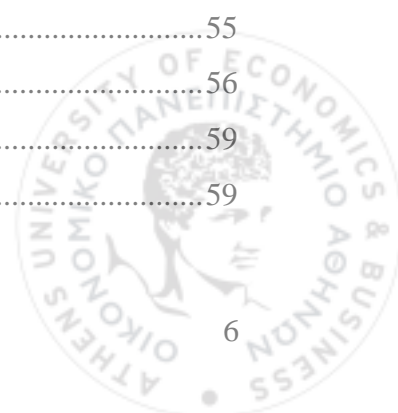
ΠΕΡΙΛΗΨΗ

Η παρούσα μεταπτυχιακή διπλωματική εργασία επικεντρώνεται στην αντιμετώπιση της δυσκολίας συνδυασμού και τυποποίησης των σχολίων μεταξύ διαφορετικών εργαλείων Επεξεργασίας Φυσικής Γλώσσας, ειδικά κατά την ανάλυση επιστημονικής βιβλιογραφίας. Η προτεινόμενη λύση βασίζεται σε μία γενικευμένη προσέγγιση αντιστοίχισης. Η προσέγγιση αυτή αντιστοιχεί τα ερευνητικά αντικείμενα σε επίπεδο πρότασης από το εργαλείο Research Artifact Analysis σε αρχεία CAS που δημιουργούνται μέσω της πλατφόρμας INCEPTION. Αυτή η ενσωμάτωση είναι κρίσιμη, καθώς αντιμετωπίζει ασυνέπειες στον διαχωρισμό προτάσεων, την κατάτμηση των λέξεων και τις μορφές κωδικοποίησης, οι οποίες αποτελούν σημαντικά εμπόδια για την αξιόπιστη αντιστοίχιση σχολίων. Για την εύρεση της πιο ακριβούς ευθυγράμμισης προτάσεων, η παρούσα έρευνα αξιοποιεί τεχνικές κανονικοποίησης κειμένου καθώς και τον αλγόριθμο Longest Common Subsequence. Για τη διασφάλιση υψηλής ακρίβειας, το πλαίσιο περιλαμβάνει επίσης μηχανισμούς που εφαρμόζουν αντιστοίχιση χωρίς διάκριση πεζών-κεφαλαίων και εναλλακτικές μεθόδους. Τα ερευνητικά αντικείμενα και τα αντίστοιχα μεταδεδομένα τους προστίθενται προγραμματιστικά στη δομή CAS. Η έρευνα αυτή συμβάλλει στην επιστημονική κοινότητα παρέχοντας ένα ευέλικτο πλαίσιο για τη βελτίωση της συνέπειας των σχολίων μεταξύ εργαλείων Επεξεργασίας Φυσικής Γλώσσας. Τα αποτελέσματά της ενισχύουν την ανάπτυξη πιο αξιόπιστων και αυτοματοποιημένων ροών εργασίας, με δυνατότητες εφαρμογής σε πολλούς επιστημονικούς τομείς.



CONTENTS

ABSTRACT.....	4
ΠΕΡΙΛΗΨΗ.....	5
CONTENTS.....	6
LIST OF FIGURES	8
CHAPTER 1	9
CHAPTER 2	16
2.1 Prodigy.....	16
2.2 Doccano	18
2.3 WebAnno.....	20
2.4 Research Artifact Analysis (RAA)	22
2.5 INCEpTION	23
2.6 Compatibility Challenges with INCEpTION	26
2.7 Sentence Alignment Methods in INCEpTION	28
2.8 Longest Common Subsequence (LCS).....	29
2.9 CAS Files in NLP Annotation Pipelines	30
2.10 Annotation Mapping Approaches	31
CHAPTER 3	37
3.1 Annotation Output from RAA	37
3.2 Output from INCEpTION	40
3.3 Code Implementation.....	44
3.4 Annotation Mapping Challenges	49
3.5 Evaluation Metrics	51
3.6 Qualitative Analysis	52
3.7 Quantitative Analysis	53
CHAPTER 4	54
4.1 CAS Typesystem Limitations.....	54
4.2 Challenges in Handling Files as JSON	55
4.3 Unresolved Issues and Future Work	56
CHAPTER 5	59
5.1 Analysis of Research Findings.....	59



5.2 Contributions to NLP Annotation.....	60
5.3 Recommendations for Future Research	61
APPENDIX.....	64
REFERENCES	84



LIST OF FIGURES

Figure 1: RAA xlsx output file - sheet 1.	38
Figure 2: RAA xlsx output file - sheet 2.	39
Figure 3: RAA xlsx output file - sheet 3.	40
Figure 4: The main dashboard of the INCEpTION annotation tool.	41
Figure 5: Document import interface in INCEpTION.	42
Figure 6: Document selection interface in INCEpTION for annotation.	43
Figure 7: INCEpTION annotation interface for manual labelling or exporting the PDF in CAS format.	43
Figure 8: Annotated research paper in INCEpTION.	49

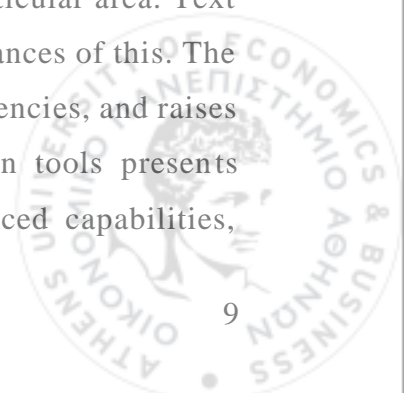


CHAPTER 1

Unstructured data is expanding at a fast rate in the digital age. This changes how information is processed and understood. Challenges in analysis and annotation arise due to the growing amount of textual data, especially in the form of academic papers, historical documents, and PDFs [1]. Even though automated tools that extract and process these data have advanced significantly, there are still basic obstacles to overcome. These difficulties are focused on the interpretation and visualization of results. The lack of a user-friendly and integrated visualization technique that enables interaction with extracted data is a major drawback. Researchers, data curators, and annotators face challenges since they have to both interpret automated outputs and improve their accuracy and usability.

Annotation and document analysis depends on a set of specialized tools. Each of these tools performs specific tasks, that include entity recognition, semantic tagging, and text extraction. Although each of these tools works well on its own, workflows are made more difficult due to their disjointed nature [2]. As a result, integration is a manual process that is prone to mistakes. Although existing annotation tools are equipped with advanced features, the annotation tools that are currently in use, lack integration mechanisms. This results in inconsistent interpretation and increased complexity in making necessary corrections.

In general terms, in Natural Language Processing (NLP), there are still many problems with the way data is integrated and visualized in analytical workflows, regardless of their advancements in text mining and automated annotation [3, 4]. Several problems still slow down the progress. To begin with, there is not a visualization mechanism that is easy to use. Information extraction tools lack an effective way to visualize results. As a result, researchers and curators find it difficult to interpret results and spot inconsistencies due to the lack of a user-friendly representation. Another problem is the complexity of PDF analysis workflows. Analyzing textual data within PDFs usually requires many independent tools. Each tool focuses on a particular area. Text extraction, semantic structuring, and metadata tagging are a few instances of this. The complex nature of these workflows adds mental load, creates inefficiencies, and raises the possibility of data loss. The integration of current annotation tools presents additional difficulties. Although annotation platforms offer advanced capabilities,

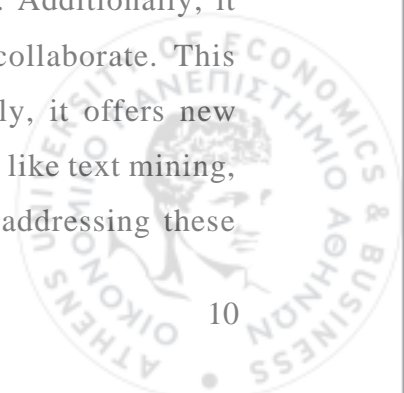


they do not have methods for combining outputs with other analysis tools. Because of this, researchers have to manually align and process annotation data, which results in inconsistencies and inefficiencies. Last but not least, a significant obstacle is the limitations of automation when there is no human oversight. Although automated systems can analyze large volumes of text, human validation and correction are necessary for their outputs. However, human intervention is tedious and difficult due to the lack of an interactive system that permits direct interaction with annotations and extracted data.

A new approach that facilitates easy tool integration, improves interpretability, and allows for real-time human oversight is needed to address these issues. A more efficient method of textual data curation can be developed by addressing the existing gaps in Artifact Analysis, PDF processing, and integration of annotation tools. The enhancement of accuracy and usability of these procedures will contribute to making research methods more reliable and efficient.

This visualization approach must be created in order to overcome the current constraints in workflows for text analysis and document annotation [1, 5]. Creating an interactive visualization technique that offers an easy way to explore extracted data is one important area of focus. Another aspect is ensuring that existing annotation tools are integrated well in order to facilitate compatibility and easy data exchange. The improvement of the interpretability with graphical representations is also beneficial since it makes it simpler to spot relationships, inconsistencies, and areas that need human intervention. Supporting a human-in-the-loop (HITL) approach is another feature that allows curators and annotators to directly interact with and improve machine-generated annotations. Last but not least, evaluating the effectiveness of the suggested system through user research and real-world case studies is another effective method to measure improvements in efficiency.

By refining visualization techniques and workflows, it aims to enhance the efficiency of annotation workflows [6]. This reduces the time and effort that is needed to manually integrate several tools. Additionally, it increases the accuracy of extracted data and enables human curators to systematically validate outputs. Additionally, it makes it easier for automated and manual curation processes to collaborate. This guarantees reliability in large-scale document analysis. Additionally, it offers new insights on visualization techniques that can be applied to other fields like text mining, computational text processing, and Artificial Intelligence (AI). By addressing these

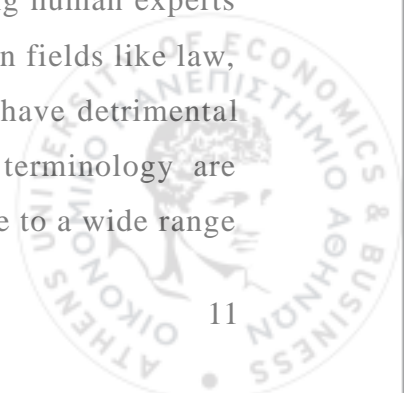


aspects, this study contributes to the continuous efforts to improve document analysis, refine annotation technologies, and enhance the cooperation between automated tools and human knowledge.

The importance of an interactive and adaptable visualization framework is becoming more evident, especially for HITL systems. Human knowledge is essential for HITL systems in order to validate and improve automated outputs. A visualization approach would give curators and annotators an easy way to interact with data. This would help with decision-making, knowledge enrichment, and error detection.

A lot of attention is given to HITL systems. The HITL concept is a computational model that incorporates human expertise into automated systems. This ensures that human intervention reviews and improves machine-generated outputs [7]. Since total automation often leads to inconsistent results and lack of contextual understanding, HITL is helpful in fields like NLP, AI, data annotation, and decision-making processes. HITL systems operate in an iterative manner. More specifically, automated processes are given feedback and corrections by humans, which enables machines to learn and adapt over time. Instead of relying on fully autonomous AI models, HITL combines the efficiency of Machine Learning (ML) with human expertise. This enhances the accuracy and reliability of AI-generated results. HITL is essential for validating extracted data in document annotation and NLP applications. It guarantees that mistakes in sentence segmentation, entity recognition, and metadata tagging are fixed prior to final use. This is particularly crucial when handling linguistic interpretations, ambiguous meanings, and complex text structures, all of which AI models may struggle to process alone.

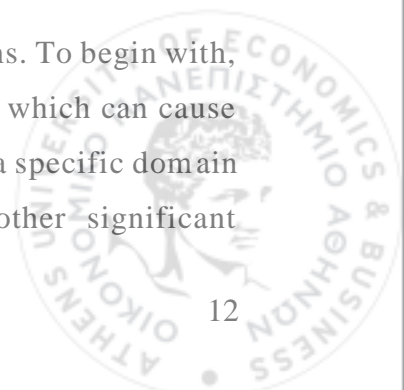
HITL provides a number of advantages in data-driven workflows, particularly in NLP, annotation tasks, automated decision-making, and AI model training. One of its main benefits is the fact that HITL increases the reliability of machine-generated outputs. Despite the advanced capabilities of AI and NLP models, they are vulnerable to biases in text processing, misclassification of entities, and interpretation errors. By enabling human reviewers to spot and address these biases in automated decisions, HITL helps reduce these problems. HITL ensures more relevant results by letting human experts validate and improve the extracted data. This is particularly crucial in fields like law, medicine, and academia where inaccurate AI-generated results can have detrimental effects. Additionally, context, cultural differences, and specific terminology are frequently difficult for AI models to comprehend. HITL is applicable to a wide range



of industries, such as academic research, healthcare, finance, legal systems, and content moderation. Because of its adaptability, HITL can be customized for particular domains and applications, which makes it an effective tool for increasing automation in a variety of fields. By using their domain knowledge, experts can refine outputs through human intervention. This may result in better interpretation of textual data. Additionally, due to the flexibility of HITL systems, AI models can be continuously trained and refined. By providing iterative feedback mechanisms, human corrections are used to improve ML models over time. This implies that they become more efficient at managing real-world situations.

Regardless of its benefits, HITL has problems and limitations that must be resolved [8]. One of its main drawbacks is the fact that HITL slows down automation workflows. Tasks that could be completely automated take longer to finish because human intervention must be provided at several points. In applications where speed is most important, this could be a challenge. Also, integrating human expertise into AI-driven systems increases costs, as organizations must employ skilled annotators, curators, and experts to review outputs. As a result, HITL is more expensive than fully automated solutions, especially for companies and research projects with budget constraints. Furthermore, although human involvement increases precision, it also raises the possibility of human error. The same content may be interpreted differently by various reviewers or annotators, which could result in inconsistent annotations or corrections. This subjectivity could decrease the overall effectiveness of HITL approach, especially when specific guidelines are not followed. Moreover, scaling HITL-based systems is a major challenge as data volume rises. Large-scale annotation, data validation, and document review require a large number of human resources, which makes it difficult to scale HITL for big data applications. Last but not least, HITL systems need constant manual input from human annotators, which over time may cause mental exhaustion and reduced productivity. Repetitive tasks such as validating entity recognition, correcting sentence segmentation, and reviewing large datasets can become overwhelming and reduce the overall productivity and accuracy of human annotators.

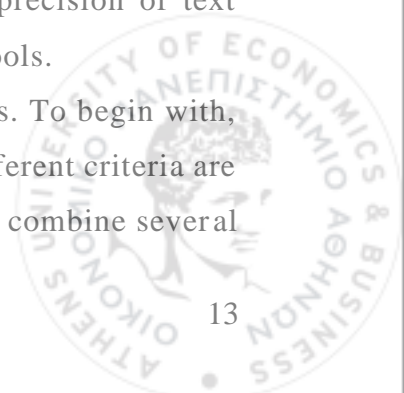
Despite its advantages and disadvantages, HITL has several limitations. To begin with, HITL systems are dependent on the availability of human reviewers, which can cause delays in workflows. If annotators are not available or if expertise in a specific domain is limited, the HITL process can become inefficient. Also, another significant



limitation of HITL is finding the perfect balance between automation and human intervention. On the one hand, over-reliance on human reviewers is in contrast to the purpose of automation. On the other hand, excessive automation can reduce accuracy. Careful system design and constant optimization are necessary to achieve a balance. Additionally, humans contribute a variety of perspectives and biases to the annotation process. This may result in inconsistent labelling and variations in decisions. In case there is a lack of standardization procedures, different reviewers may make inconsistent corrections or annotations, which may compromise the overall quality of the dataset. Furthermore, sensitive data in industries such as healthcare, finance, and legal services, is frequently processed by HITL systems. The involvement of human reviewers raises privacy concerns, especially when dealing with confidential or personal information. Data security procedures must be implemented by organizations to stop leaks of information and guarantee compliance to laws like GDPR. Although HITL enables AI models to learn from human corrections, it can be difficult to set up efficient iterative feedback mechanisms. It takes a lot of computational strength and ML knowledge to include annotator corrections into AI model retraining procedures. Nevertheless, HITL systems serve as a crucial bridge between automation and human expertise, ensuring higher accuracy, better interpretability, and improved error handling. Although HITL contributes to several fields, it additionally raises challenges with cost, processing time, scalability, and human variability. For applications where precision, contextual awareness, and human oversight are critical, HITL is a crucial approach despite its drawbacks. In order to optimize the advantages of such an approach but minimize its limitations, more effective models may be developed by both humans and AI, along with interactive visualization tools.

As mentioned before, there are critical challenges in NLP-based text processing, when integrating outputs from different annotation tools. One of the most crucial challenges is that there is a lack of alignment between sentences. There are inconsistencies in text alignment and annotation placement because different tools, like INCEpTION and Research Artifact Analysis (RAA), apply different rules for sentence segmentation and encoding. These variations reduce the overall efficiency and precision of text analysis workflows and make it more difficult to integrate various tools.

There are several factors that cause misalignment in annotation tools. To begin with, one of the main causes is the variation in sentence segmentation. Different criteria are used by different NLP tools for splitting sentences. While some may combine several

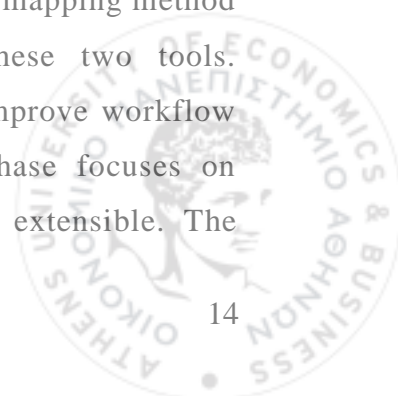


clauses into a single segment, others may handle complex sentences as distinct units. Some tools use syntactic analysis to identify logical sentence boundaries, while others rely on strict punctuation rules, such as breaking at every period, comma, or semicolon. Also, there are text encoding differences, such as Unicode variations, character encoding mismatches, or inconsistent handling of special characters, that further destroy alignment. Moreover, some tools generate structured annotation formats, such as XML, JSON, CoNLL, while others produce flat text as an output. This makes direct integration really challenging. These inconsistencies make it difficult to merge, compare, and refine annotated texts and reduce the efficiency of annotation processes [9].

In order to address the misalignment issues that are caused by these differences, a mapping-based method is suggested for development [10]. This method aims to standardize and align textual outputs across various NLP annotation tools, in order to maintain consistency of annotations across various systems. The main purpose of this solution is to develop a mapping framework that can be used with a variety of NLP tools. The mapping approach should be scalable and generalized and should enable future adaptation to other annotation platforms.

The features of this approach include several aspects. To be more precise, the mapping system is independent of the sentence segmentation logic of any particular tool. Instead, it has alignment principles that are applicable in various annotation contexts. The mapping system normalizes these formats to ensure compatibility because different NLP tools produce outputs in different formats. Also, the approach incorporates functions that automatically adjust annotations by detecting sentence mismatches, encoding inconsistencies, and variation in text representations. The mapping system is scalable and effective for developing annotation technologies. This is based on the fact that it is designed as a generalized framework that guarantees future NLP tools can be integrated without requiring significant modifications.

As a proof of concept, the suggested mapping system is first tested with INCEpTION and RAA, two widely used annotation tools. There are a variety of aspects that are checked during this testing phase. These include confirming that the mapping method is successful in dealing with segmentation errors between these two tools. Additionally, it examines how integrating several NLP tools can improve workflow efficiency and consistency of annotations. Although the first phase focuses on INCEpTION and RAA, the mapping solution is designed to be extensible. The



approach can be modified to work with other NLP tools, making it a solution for the annotation integration of several tools. By employing this mapping-based method, the study attempts to provide an automated and flexible way to enhance integration across various NLP annotation platforms.

The purpose of this master thesis is to develop a generalized mapping-based method for aligning annotations across different NLP tools. This mapping method is specifically tested using the INCEpTION and RAA tools. The goal of this mapping method is to address sentence segmentation mismatches, encoding inconsistencies, and variations in structure in order to enhance annotation integration and workflow consistency. This thesis is organized as follows. The purpose of Chapter 2 is to provide an overview of several NLP annotation tools, challenges in sentence alignment, key computational methods and the role of CAS file structures in annotation pipelines. The purpose of Chapter 3 is to describe the experimental setup, the implementation details of the proposed mapping approach, and a qualitative and quantitative analysis of the results. This provides insights into the effectiveness and limitations of the method. The purpose of Chapter 4 is to analyze the main challenges that were met during the implementation of the mapping approach and highlight unresolved problems. Last but not least, the purpose of Chapter 5 is to summarize the key findings from the analysis and propose potential future research directions for expanding the applicability of this method.



CHAPTER 2

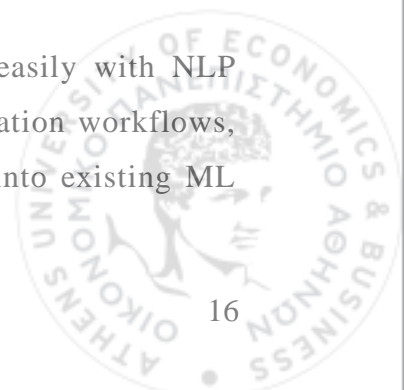
Annotation tools are essential for training AI models across multiple domains. NLP annotation tools such as INCEpTION, Prodigy, WebAnno, and Doccano can create datasets for entity recognition, text classification, and syntactic parsing. Biomedical and legal annotation tools such as MedTator, Tagtog, and BioC Annotation Tool are used for medical and regulatory text processing. Image and video annotation tools such as CVAT, Labelling, and Supervisely are used in computer vision applications, while audio annotation tools like Praat, ELAN, and Audacity support speech recognition and linguistic research. Each of these tools guarantee high-quality labelled datasets, which enhances the performance and reliability of AI systems. The main focus of this study is on NLP annotation tools, which are briefly described below.

2.1 Prodigy

Prodigy is a Python-based annotation tool that has been created by Explosion AI [11]. Its purpose is to enhance the annotation process for ML and NLP applications as it combines active learning and labelling with the ML. Unlike traditional manual annotation tools, Prodigy allows users to annotate by utilizing AI predictions and to modify annotations in real time. This makes it an excellent solution for developers to create high-quality training datasets for NLP models.

Prodigy is well-known for its interactive and scriptable environment. One of its primary features is active learning, which allows the tool to continuously learn from user inputs and prioritizes the most uncertain cases for annotation. This reduces the time that is required for manual labelling, as annotators concentrate only on the most useful information. Furthermore, NER, text classification, dependency parsing, relation extraction, and image and speech annotation are among the various annotation types that it supports.

Additionally, Prodigy is completely customizable and integrates easily with NLP frameworks. Due to its adaptability, users can create custom annotation workflows, define labelling frameworks for specific domains and integrate it into existing ML



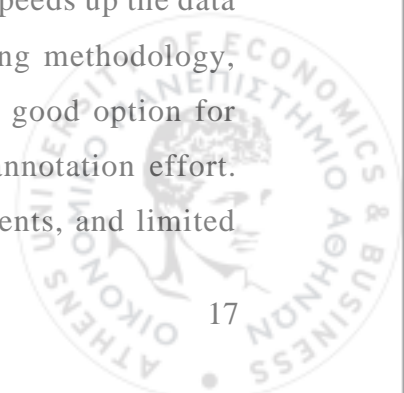
pipelines. The tool makes it simple to export and process datasets for model training by storing annotated data in structured formats like JSONL.

Prodigy is generally used for various NLP and ML applications, especially when it comes to data annotation for supervised learning. It is used in text classification, where it helps categorize large datasets into predefined labels, and in NER, where it helps to identify particular entities. Prodigy is also utilized in relation extraction where it helps to find relationships between textual entities, and in dependency parsing which includes examining the grammatical structure of sentences. In addition to NLP, Prodigy allows image annotation, which makes bounding box labelling possible for object detection tasks. Additionally, it has been used in speech and audio annotation, where it helps in speech recognition model training. Due to its adaptability, it is useful in many sectors where AI models require high-quality labelled datasets.

One of the main advantages of Prodigy is its efficiency in eliminating annotation time while preserving high-quality training data. By utilizing AI-powered annotation, Prodigy reduces the manual effort that is needed, which lets annotators focus only on the most important cases. It is also flexible for various ML tasks due to its easy integration with ML frameworks. Additionally, the tool offers structured data storage by storing annotated outputs in JSONL format. This facilitates easy integration with NLP pipelines and model training workflows.

Prodigy has certain drawbacks in spite of its benefits [12]. In contrast to open-source annotation tools, Prodigy is a commercial tool that requires a paid license, which could be a constraint for developers or companies with limited budgets. Additionally, it needs programming knowledge, as users need to be familiar with Python scripting in order to execute annotation workflows. This requirement could be difficult to meet for non-technical users who look for an efficient annotation solution. Another drawback is its lack of internal multi-user collaboration features. In contrast to tools that facilitate collaborative annotation projects, Prodigy is primarily intended for single users or small teams. Because of this, it is less appropriate for large dataset curation projects that need several annotators to work simultaneously.

In conclusion, Prodigy is an extremely efficient annotation tool that speeds up the data labelling procedure for NLP and ML applications. Its active learning methodology, customizability and easy integration with ML pipelines, makes it a good option for researchers who want to develop high-quality datasets with little annotation effort. However, its commercial nature, programming knowledge requirements, and limited



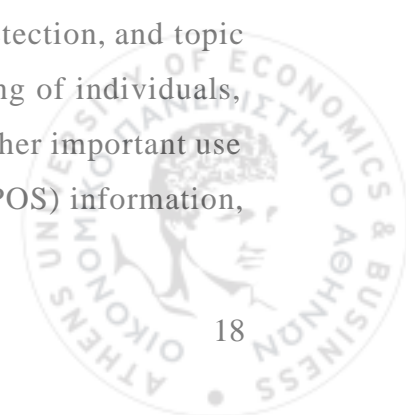
collaborative capabilities might make it complex for some users. Nevertheless, Prodigy continues to be a useful tool for individuals and groups that are involved in NLP research and model development, which serves as a bridge between human expertise and machine automation in data annotation.

2.2 Doccano

Doccano is an open-source, web-based annotation tool that is made specifically for NLP tasks and enables users to efficiently produce labelled datasets for ML applications. Unlike commercial annotation tools that need paid licenses, Doccano can be used for free and is accessible to both researchers and developers. Its simple interface makes it possible for users to annotate text for a variety of purposes. Since it is a web-based platform, Doccano allows for team-based annotation, which makes it a useful tool for creating large-scale datasets.

Doccano is designed to be both user-friendly and efficient, with its drag-and-drop interface that makes annotation tasks easier. One of its important features is multi-user collaboration, which enables multiple annotators to work on the same dataset in real-time. For research teams and businesses that work on large NLP projects, this makes it really helpful. Additionally, it supports a wide range of annotation types, such as text classification, entity recognition, sequence labelling, and relation annotation. In order to ensure compatibility with ML frameworks, users can simply highlight text, assign labels, and export annotated datasets in a variety of formats, including JSON, CSV, and plain text. Furthermore, it has internal support for multilingual text annotation, which makes it useful for annotating datasets in multiple languages. Due to the fact that it is an open-source tool, developers can change its code to suit particular annotation requirements.

Doccano is a popular tool for creating NLP datasets for use in business applications, academic research, and AI development. It is useful for text classification, where users group documents into categories, such as sentiment analysis, spam detection, and topic classification. It is also utilized for NER, and it helps in the labelling of individuals, places, organizations, and other named entities in text [13, 14]. Another important use case for Doccano is sequence labelling, which tags part-of-speech (POS) information,

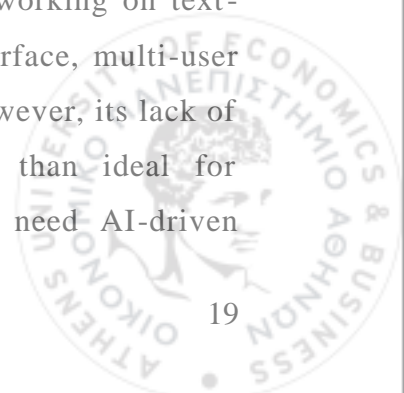


syntactic dependencies, and tokenized structures. Furthermore, it facilitates relation annotation, which allows users to label connections between various text elements, such as identifying relationships between individuals in a document. Due to its multi-user collaboration features, Doccano is frequently used in collaborative NLP projects that need collaborative annotations or large-scale data labelling.

One of the greatest benefits of Doccano is that it is free and open source, which makes it a great option for those with limited budgets. In contrast to commercial tools like Prodigy that require a paid license, Doccano is an open-source platform that provides full feature access without charging for licenses. Another significant benefit is its ease of use. Due to its simple web-based interface, users can begin annotating without the need for technical knowledge. Doccano makes it simple for non-technical users to annotate data without the need for programming knowledge, in contrast to Prodigy, which requires Python scripting. Additionally, Doccano facilitates multi-user collaboration, which makes it scalable for large teams. Users can efficiently assign annotation tasks, review labelled data and manage annotation projects. It is a flexible tool since it can export datasets in a variety of formats, guaranteeing compatibility with different ML models.

Doccano has certain drawbacks in spite of its benefits. One of its main disadvantages is its lack of advanced automation features. Doccano only uses manual annotation, which can be time-consuming for large datasets, in contrast to Prodigy, which offers active learning and AI-driven annotation. Another limitation is its limited ability to support complex annotation workflows. Although it works well for simple text classification and entity recognition tasks, it does not support dependency parsing, hierarchical annotations, and structured data relationships, which are features that other tools offer. Furthermore, although Doccano facilitates multi-user collaboration, it lacks role-based access control and quality assurance features. These are crucial for large-scale annotation projects that require multiple reviewers and checks on annotation consistency [15].

Doccano is a robust and easy open-source annotation tool that provides a great option for creating NLP datasets. Developers and organizations that are working on text-based ML models prefer this tool because of its user-friendly interface, multi-user collaboration features, and support for various annotation types. However, its lack of automation and advanced annotation capabilities make it less than ideal for complicated NLP tasks or large-scale commercial projects that need AI-driven



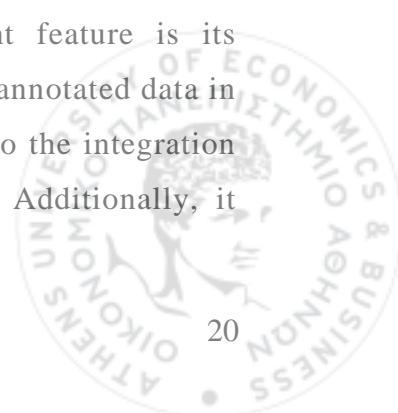
labelling. Nonetheless, Doccano continues to be one of the greatest options for groups that search for an affordable web-based annotation tool with strong collaboration capabilities.

2.3 WebAnno

WebAnno is an open-source, web-based annotation tool that was created for use in linguistic and NLP applications. Created by the Technische Universität Darmstadt, WebAnno is a collaborative environment where multiple users can annotate text data for ML models and linguistic research. In contrast to independent tools that need local installations, WebAnno operates on a server-based architecture and enables collaborative annotation while preserving centralized access to data [16].

WebAnno is helpful for tasks such as NER, POS tagging, dependency parsing, and relation annotation. Due to its support on annotation frameworks that are defined by users, it is adaptable for applications in specific domains, such as biomedical text annotation, legal document analysis, and social media text processing. The tool also ensures compatibility with a variety of NLP frameworks and ML pipelines by integrating with the Common Analysis Structure (CAS) and Unstructured Information Management Architecture (UIMA) files.

One of its primary strengths is its collaboration feature, which enables several annotators to work on the same project simultaneously. In contrast to tools such as Prodigy or Doccano, which are designed primarily for individual or small-team annotation, WebAnno is built for large-scale collaborative projects. Role-based access control is one feature that makes it possible to assign particular permissions. WebAnno supports various annotation types, including text classification, sequence labelling, coreference resolution, and syntactic parsing. Users can define word relationships, add labels, highlight text, and create hierarchical annotations using the interface. It is especially helpful for linguistic analysis and dataset construction because it also offers visual representations of syntactic structures. Another important feature is its compatibility with external NLP tools. Users can import and export annotated data in a variety of formats, such as JSON, XML, CSV, and CoNLL, due to the integration with UIMA and other NLP frameworks that WebAnno supports. Additionally, it

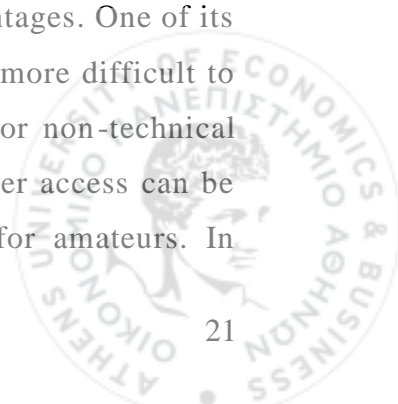


provides automated pre-annotation features that let users train models and apply annotations automatically, which can then be fixed manually.

WebAnno is frequently used in academic research, NLP development, and industry applications where structured text annotation is needed. It is employed to construct linguistic datasets, allowing researchers to produce high-quality datasets for NLP model training. One of its primary uses is NER, where it helps classify entities such as names, locations, and dates. It is also widely used for POS tagging, dependency parsing, and syntactic structure analysis, which make it an excellent tool for computational linguistics and syntax-related NLP tasks. Relation extraction and coreference resolution are two more significant uses for WebAnno, where it is employed to track references between sentences and annotate the relationships between entities. This makes it especially helpful for legal document processing, biomedical text mining, and social media sentiment analysis. Due to its collaborative features, it is also utilized in large-scale annotation projects where several annotators must cooperate to guarantee data consistency and quality control.

One of the biggest advantages of WebAnno is that it facilitates extensive and collaborative annotation projects [13, 17]. In contrast to simpler annotation tools, it enables multiple users to work on the same dataset at parallel, thus making it perfect for enterprise or research-level projects. Another significant benefit is its adaptability. Since it enables users to define custom annotation schemas, it can be tailored for a wide range of NLP tasks, from simple text classification to complex syntactic parsing and relation extraction. Its pre-annotation features also increase efficiency. WebAnno can automatically annotate text by integrating with ML models, so that human annotators can then review and edit the results. This reduces the amount of manual effort that is required and accelerates the data labelling process. Additionally, WebAnno supports a variety of export formats, making it compatible with ML frameworks, linguistic tools, and NLP pipelines. Annotated data can be reused in a variety of computational linguistics applications thanks to its integration with UIMA and DKPro.

WebAnno has a few significant drawbacks in spite of its many advantages. One of its main drawbacks is its requirement for server setup, which makes it more difficult to install and use than independent tools like Prodigy or Doccano. For non-technical users, configuring database connections, user permissions, and server access can be challenging. Another drawback is its greater level of difficulty for amateurs. In



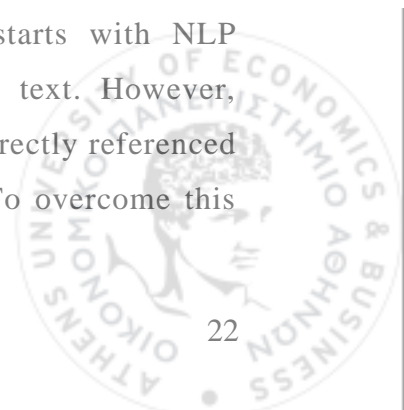
contrast to more straightforward annotation tools that provide drag-and-drop interfaces, WebAnno is less usable by non-expert annotators due to its need for knowledge of annotation structures and linguistic concepts. Its automation features are not as sophisticated as those of Prodigy, which optimizes annotation workflows through active learning, despite supporting pre-annotation with ML models. Its ML integration is manually configured and does not change in response to user input. Furthermore, it does not provide real-time synchronization for very large datasets, which means that several annotators that are working on the same document might deal with versioning conflicts. For projects that require hundreds of annotators to work concurrently, this could be a drawback.

Nevertheless, WebAnno is a robust, open-source annotation tool that works well for collaborative NLP annotation projects. Its multi-user collaboration features, adaptable annotation schemes, and compatibility with NLP frameworks make it a perfect option for research institutions and academic projects. However, its need for a server setup, greater level of expertise, and limited real-time synchronization for large datasets might be an obstacle for users who are looking for a more straightforward annotation process. Despite these limitations, it is still one of the most popular annotation platforms for linguistic and NLP research, which makes it a useful tool for producing high-quality, annotated datasets for ML and computational linguistics applications.

2.4 Research Artifact Analysis (RAA)

RAA is a field that is focused on the identification and categorization of research artifacts that are referenced in scientific literature [18, 19]. These artifacts play a crucial role in the research process as they ensure that scientific findings are both reproducible and reusable. The main objective of RAA is to analyze and structure the data of these artifacts to enhance knowledge discovery.

RAA combines text mining, ML, and annotation techniques to automatically extract research artifacts from academic papers. The process usually starts with NLP techniques, such as NER, which finds references of artifacts in text. However, traditional NER models find difficulty in capturing unnamed or indirectly referenced artifacts. As a result, the information that is extracted has gaps. To overcome this



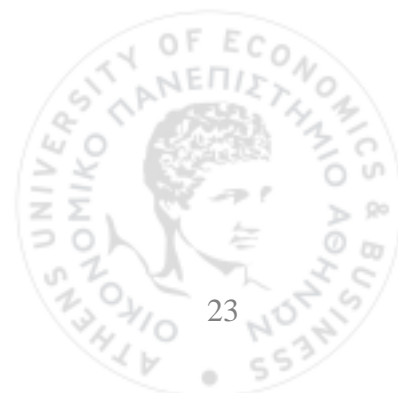
restriction, recent developments in RAA incorporate LLMs that are optimized with particular datasets. By employing guided questioning techniques, these models are able to identify research artifacts even when they are not specifically stated in the text. An essential part of RAA is annotating the extracted artifacts. Each identified mention consists of some metadata such as versioning, licensing, and URLs, and provides details about the use and availability of the artifact. This metadata is crucial for linking artifacts across publications and enabling researchers to access computational resources efficiently.

One of the main benefits of RAA is its ability to automate the process of manually locating research artifacts in vast volumes of literature. By utilizing ML models and annotation frameworks, RAA accelerates the extraction and organization of scientific resources. This supports open science practices, by making it simple for researchers to locate and utilize datasets and software that are created by others. Moreover, structured artifact analysis enhances citation tracking and impact assessment, as it makes it possible to assess how research artifacts are used in various studies.

Despite its benefits, RAA has a number of drawbacks. One of the main challenges is the variation in how research artifacts are referenced in academic papers. While some artifacts are clearly identified by their names, others are only mentioned indirectly, which makes automated detection challenging. Furthermore, variations in writing styles bring inconsistencies that make extraction tasks more difficult. Another drawback is its requirement for high-quality training data for ML models. Since current datasets concentrate only on explicitly named artifacts, there is a requirement for more datasets that record a greater range of artifact mentions, including indirect references. Additionally, annotation processes can take a lot of time and may need more resources.

All things considered, RAA marks an important breakthrough in the analysis of research artifacts, as it improves the accessibility of scientific resources. By employing NLP techniques, ML models, and structured metadata annotation, RAA improves the ability to track and utilize datasets and software across multiple domains.

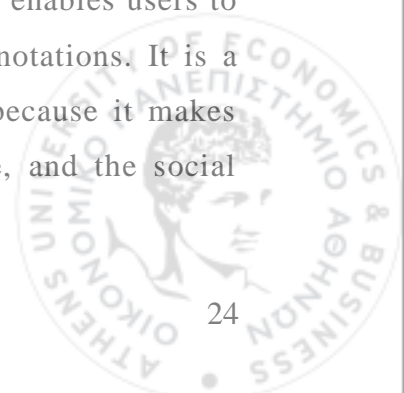
2.5 INCE_pTION



INCEpTION is an open-source, web-based annotation tool that was created to make text annotation, dataset construction, and NLP model training easier [20]. It was also created by the Technische Universität Darmstadt and is an extension of WebAnno, where features were added to improve the annotation process. INCEpTION is well-known for its ability to incorporate ML models into the annotation procedure and enable semi-automated labelling via active learning and pre-annotation capabilities. This feature reduces the workload for human annotators by recommending labels that can be verified or corrected, while increasing both the speed and accuracy of dataset creation. In contrast to many independent annotation tools, INCEpTION offers a collaborative environment which allows multiple users to work in parallel, ensures consistency across annotations and enables large-scale annotation projects. This study highlights this particular NLP annotation tool, since the experimental analysis in Chapter 3 is primarily carried out using this.

One important characteristic of INCEpTION is that it supports multi-document annotation, which enables annotators to connect related entities and concepts across several texts. This functionality is important for applications that need entity linking, semantic search, and relation extraction, where consistency across documents is essential. Additionally, it connects to external knowledge bases such as Wikidata. This makes it possible to match structured knowledge repositories with annotations. The tool supports multi-layer annotation, which enables users to work on several linguistic levels, such as syntactic structures, named entities, and relations within the same interface. This adaptability makes it a powerful tool for linguistic research, biomedical text mining, and legal document processing, where domain-specific annotation requirements are necessary.

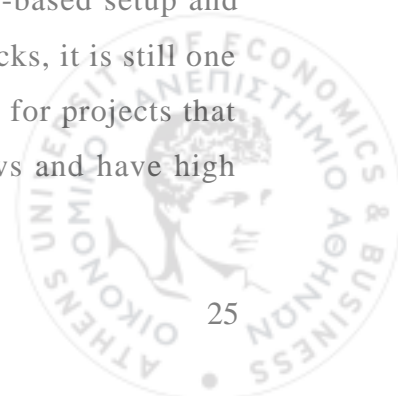
INCEpTION is extensively used in many NLP applications, such as NER, relation extraction, coreference resolution, semantic role labelling, and dependency parsing. It can handle complicated NLP tasks in addition to basic text classification. These tasks include conducting in-depth syntactic analysis, determining relationships between entities across documents, and connecting annotations to knowledge bases. Its efficiency is increased by integrating AI-assisted annotation, which enables users to automate tasks and concentrate on improving model-generated annotations. It is a useful tool for researchers who work with large textual datasets because it makes structured data extraction easier in fields like healthcare, finance, and the social sciences.



One of the main benefits of INCEpTION is its AI-assisted annotation, which significantly reduces the time and effort that are required for manual data labelling [21, 22]. By utilizing pre-trained models and active learning methods, the system prioritizes uncertain cases, which allows human annotators to concentrate on the most important corrections. This method accelerates the annotation process and improves model training with the integration of human expertise into iterative learning processes. Another significant benefit is its ability to facilitate multi-user collaboration, which makes it suitable for research teams and organizations that work on large-scale NLP projects. Also, it provides customization options, which let users create their own annotation schemas, adapt the tool to specific domain requirements, and incorporate it into existing NLP pipelines. It also offers a variety of export choices, supporting multiple formats such as JSON, XML, CoNLL, and RDF, which means that it is compatible with a large number of ML frameworks.

In spite of its advantages, INCEpTION has certain disadvantages. One of the main obstacles is its complicated installation procedure, which necessitates a server setup and database configuration. In contrast to simpler annotation tools such as Doccano, which require little setup, it requires technical expertise to deploy. Another limitation is its requirement for technical expertise. Due to its sophisticated features and multi-layer annotation system, it may be difficult for users who are unfamiliar with linguistic annotation or NLP workflows. Furthermore, it requires a lot of computation when working with large datasets, which makes it more difficult for businesses without access to powerful computer infrastructure. Furthermore, it gives more emphasis on text annotation but lacks native support for image or speech data, limiting its applicability in AI research.

In conclusion, INCEpTION is a powerful annotation platform that combines AI-assisted learning with collaborative annotation features. This makes it a great option for NLP researchers and data scientists. It stands out from other annotation tools due to its capacity to manage complex annotation tasks, facilitate cross-document linking, and incorporate external knowledge bases. However, users without a strong background in computational linguistics or NLP may find its server-based setup and technical requirements difficult to understand. Despite these drawbacks, it is still one of the most advanced annotation platforms on the market, especially for projects that need to integrate with semantic relationship tracking, ML workflows and have high annotation accuracy.

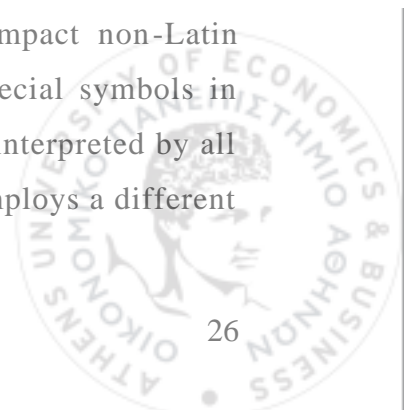


2.6 Compatibility Challenges with INCEpTION

One of the major challenges in using INCEpTION as an NLP annotation tool is to ensure sentence alignment across various tools and processing pipelines. Similar to this instance, inconsistencies occur when INCEpTION is combined with other NLP frameworks because of differences in tokenization and parsing techniques, encoding inconsistencies, and sentence segmentation variations. These inconsistencies make it difficult to maintain annotations and ensure compatibility between tools in multi-platform NLP workflows.

Sentence splitting is an essential part of NLP preprocessing, since different annotation tools and NLP frameworks use different segmentation techniques. While some tools use linguistic heuristics, ML models, or syntactic structures to identify sentence boundaries, others rely on sentence boundary detection. INCEpTION lets users personalize sentence segmentation but does not always align with the segmentation methods that are used by other platforms. As a result, sentences may not line up precisely when importing text from various tools. This leads to annotation mismatches in tasks like dependency parsing, entity recognition, and relation extraction. Furthermore, when comparing datasets, some tools may produce inconsistent sentence boundaries because they segment sentences according to fixed character limits. Different annotation tools may split a sentence differently, which can cause problems with entity linking, sentence mapping, and model training because the annotated segments may not match exactly.

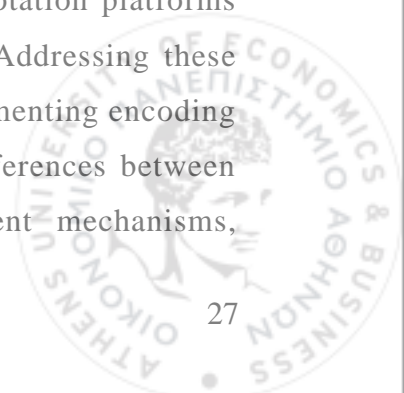
Another major obstacle in sentence alignment is represented by encoding discrepancies, which arise when different annotation tools employ different character encoding standards [23]. Although INCEpTION supports UTF-8 encoding, some legacy systems or datasets may use ISO-8859-1, ASCII, or other encoding formats. When importing and exporting annotated text, this can result in segmentation errors, character mismatches, or corrupted text. Encoding mismatches impact non-Latin scripts, special characters, and diacritical marks. For example, special symbols in some biomedical or multilingual datasets might not be consistently interpreted by all tools. When INCEpTION imports text from another platform that employs a different



encoding scheme, special characters may be misrepresented, which leads to misaligned annotations or corrupted entity labels. Another encoding discrepancy involves handling line breaks and whitespace. Line breaks are ignored by some annotation tools, while others treat them as sentence boundaries. Furthermore, invisible Unicode characters may affect sentence processing and result in alignment errors. These encoding issues can interfere with tokenization, sentence boundary detection, and word alignment, complicating compatibility between INCEpTION and other NLP tools.

Tokenization, which is the process of dividing text into words or tokens, is different for each NLP framework. INCEpTION employs customizable tokenization methods, while different tools implement their own word segmentation rules. This leads to inconsistencies in sentence alignment. While some NLP systems employ ML-based techniques that adjust to language-specific features, others perform rule-based tokenization. When different tools tokenize compound phrases, hyphenated words, or contractions in different ways, a problem occurs. These differences affect entity recognition, POS tagging, and dependency parsing, which makes it challenging to align sentence annotations across various tools. Another challenge in tokenization is in multilingual datasets where tokenization rules differ based on language structure. In some languages, like Chinese, Japanese, and Thai, where words are not explicitly separated by spaces, different NLP frameworks employ statistical models or predefined dictionaries for tokenization [24]. This causes inconsistent word boundaries, which leads to mismatched annotations when transferring labelled text between INCEpTION and other NLP platforms. Additionally, different syntactic and morphological analysis techniques lead to different sentence parsing methodologies. When text is imported into INCEpTION from tools that use different preprocessing methods, the structure of sentences can alter. This can cause misalignment in entity labels, dependency relations, and semantic roles.

Sentence alignment difficulties in INCEpTION arise due to variations in segmentation techniques, encoding differences, and diverse tokenization strategies across NLP tools. These differences complicate the compatibility between annotation platforms and need extra preprocessing procedures to guarantee alignment. Addressing these challenges involves standardizing text processing workflows, implementing encoding normalization, and creating mapping techniques that eliminate differences between annotation frameworks. In order to improve sentence alignment mechanisms,



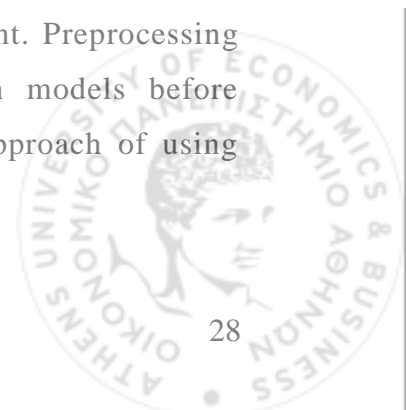
INCEpTION can improve annotation consistency that facilitates more effective integration with other NLP tools and research pipelines.

2.7 Sentence Alignment Methods in INCEpTION

Ensuring sentence alignment consistency in INCEpTION when integrating it with other NLP tools needs implementing preprocessing techniques, encoding normalization, and tokenization alignment strategies. To address differences in sentence splitting methods, custom segmentation rules are implemented within INCEpTION, in order to bring them into line with external NLP tools [25]. One method is to preprocess text using universal sentence boundary detection libraries, before importing it into INCEpTION. Sentence boundaries can also be kept consistent across frameworks by using AI-driven segmentation models and establishing consistent linguistic rules.

For encoding discrepancies, INCEpTION supports UTF-8 encoding, but compatibility problems may be caused when importing text from tools that use other encoding formats. In order to reduce this, users should normalize encoding formats prior to importing text, using preprocessing pipelines. These pipelines would handle invisible Unicode characters, detect and convert character sets, and guarantee consistent processing of whitespace. Standardizing text formatting and escaping special characters before annotation also avoid alignment errors that are brought on by encoding mismatches [23].

To address tokenization and sentence parsing inconsistencies, INCEpTION provides adjustable tokenization settings. However, when integrating with external NLP frameworks that use different word segmentation rules, alignment problems may occur. A workable solution is to use consistent tokenization techniques, which provide a consistent token segmentation approach across different tools. Furthermore, cross-tool mapping algorithms can be used to modify tokenized text, especially when contractions, hyphenated words, and multilingual scripts are present. Preprocessing text using custom Python scripts or incorporating tokenization models before annotation in INCEpTION can further improve alignment. The approach of using Python scripts is used in this study.



INCEpTION can enhance annotation consistency by using unified tokenization techniques, encoding normalization, and sentence segmentation standardization. This ensures easy integration with other NLP annotation platforms and reduces errors in tasks involving entity recognition, relation extraction, and dependency parsing.

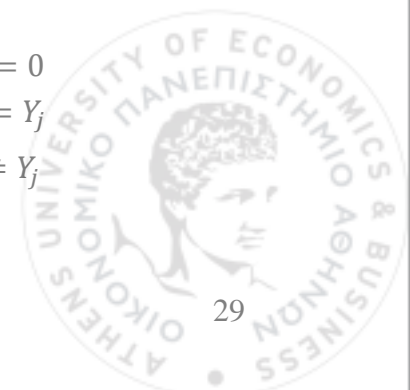
2.8 Longest Common Subsequence (LCS)

In order to ensure sentence and annotation alignment, Longest Common Subsequence (LCS) is utilized. LCS is a useful tool that can be used to identify structural similarities between sentence versions that are generated by different NLP tools. LCS helps in locating matching tokens or sequences that are consistent across versions when different annotation platforms segment text in different ways. This is the approach that is used for sentence mapping in this study.

The LCS is a widely recognized algorithm in computer science, especially in text processing and NLP applications for sequence matching, alignment, and error correction [25, 26]. As previously stated, in NLP annotation tasks, variations in sentence segmentation, tokenization, and parsing methodologies across different tools frequently result in misalignment problems when incorporating annotated datasets. By determining the longest matching sequence between two texts, LCS effectively addresses this issue and guarantees consistency in sentence mapping and annotation placement.

Mathematically, the LCS of two sequences is known as the longest subsequence that occurs in both sequences in the same order but not necessarily consecutively. The LCS is the longest sequence Z such that Z is a subsequence of both X and Y given two sequences, X of length m and Y of length n . By comparing each character and determining whether or not characters at a specific position in both sequences match, the LCS is calculated using a dynamic programming technique. The relation for LCS is given as:

$$L(X_i, Y_j) = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ L(X_{i-1}, Y_{j-1}) + 1, & \text{if } X_i = Y_j \\ \max(L(X_{i-1}, Y_j), L(X_i, Y_{j-1})), & \text{if } X_i \neq Y_j \end{cases}$$



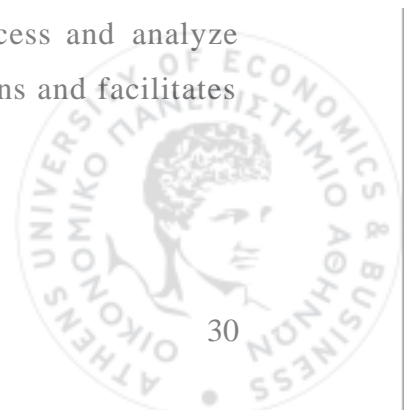
This formula enables LCS to iteratively compute the longest common subsequence, allowing to identify overlapping structures between two sequences.

Despite its benefits, LCS computation is intrinsic cost, with a time complexity of $O(mn)$, which makes it unsuitable for long text sequences. To maximize its performance, several methods have been developed. Space-efficient LCS algorithms lowers memory usage by storing only a part of the LCS table at a time, while Hirschberg's algorithm increases efficiency by employing a partitioning method that preserves time efficiency while lowering space complexity. Optimized operations can further increase LCS computation speed by processing digitally encoded sequences and applying heuristic methods such as greedy approximation algorithms. These offer nearly ideal solutions for large datasets where precise computation is computationally overwhelming. In this study, time complexity is not a relevant factor.

By implementing LCS, NLP systems can increase annotation consistency, sentence alignment, and guarantee better integration between various annotation platforms. Its ability to identify structural similarities and deal with segmentation variations makes it a crucial tool for preserving consistency in NLP workflows that incorporate many tools. Applying LCS-based mapping techniques not only increase annotation accuracy but also make it easier to process NLP data, which contribute to reliability and compatibility of annotated datasets.

2.9 CAS Files in NLP Annotation Pipelines

In order to maintain compatibility across various NLP frameworks, CAS files are an excellent choice. This happens because they improve the effectiveness of text analysis workflows by offering an organized method for managing annotations. CAS is an essential framework that is used in NLP pipelines for managing and storing annotations, sentences, along with their metadata efficiently [27]. CAS is a component of the UIMA framework that offers a methodical approach to process and analyze unstructured text data. It acts as a container for managing annotations and facilitates compatibility across various NLP tools and frameworks.



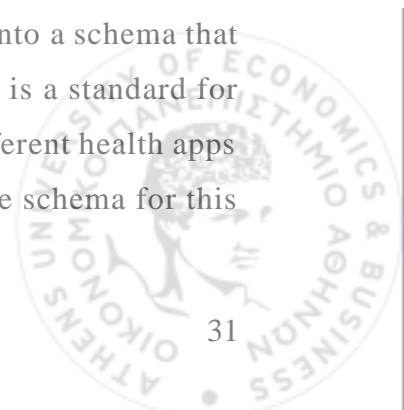
CAS files store textual data along with its annotations in an organized manner. Some examples of these annotations are named entities, syntactic structures, tokenized words, and dependency relations. This makes CAS an essential part in NLP workflows. In order to keep data traceable and reusable, CAS files also preserve metadata, including document identifiers, annotation timestamps, and processing history. This structured format enables NLP models to apply sequential text processing and preserve annotation consistency at the same time.

CAS files are used by various NLP annotation tools, like INCEpTION, to optimize their annotation processes. These tools use CAS to manage multi-layered annotations, which enable users to label text at various linguistic levels, such as entity recognition, relation extraction, and coreference resolution. Researchers can integrate different NLP models without compromising data consistency because of CAS files, which make it easier for these tools to exchange annotations.

2.10 Annotation Mapping Approaches

Research has been done in the field of NLP annotation mapping to improve compatibility between different annotation tools. It should be noted that annotation mapping is the process of developing rules or methods for transforming annotations from one semantic schema to another so that services can understand each other's data and collaborate. Several studies examined how to guarantee annotation consistency across different NLP platforms. In general, while previous research has established the foundation for annotation mapping techniques, more automated solutions are still required to improve annotation consistency across NLP workflows.

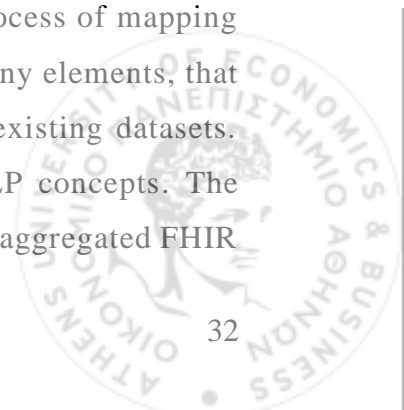
To begin with, a lot of research has been done by Hong Na, et al. [28]. The primary goal of the research is to standardize heterogeneous annotation datasets for reuse and integration into NLP for clinical data. Additionally, annotation mapping is also performed with a few simple steps. More precisely, some tools are developed that can automatically convert the annotation schema of any existing dataset into a schema that is based on Fast Healthcare Interoperability Resources (FHIR). This is a standard for exchanging healthcare data, like a common language that enables different health apps and systems communicate with each other. The is utilized as the base schema for this



conversion. This indicates that the research deals with determining how categories and annotation structures can be represented in each different dataset using the standard FHIR categories and structures. The original annotation schemas are frequently different. This means that researchers manually develop mapping rules between the original schemas and the standardized FHIR schema. These mappings are done at two levels. The first level includes mapping all sections of the original annotation data to specific FHIR “resources”. The second level includes mapping specific components within these resources. Additionally, content normalization is also performed. This research enables the integration and reuse of different annotation datasets for training and assessing clinical NLP tools.

There are several results that come from this study. To begin with, researchers were able to develop tools that can take the different schemas that are used in various annotated clinical texts and convert them automatically into a common schema that is based on FHIR. Furthermore, they developed some rules for mapping annotation concepts and categories from the original datasets to FHIR concepts and categories. This was important because the original datasets were made for different purposes and used different labels for the textual information. Furthermore, a system was developed where researchers can automatically review transformed annotations and add new ones, with common annotation tools. This guarantees quality of the standardized data. Also, the study demonstrated that it is feasible to use FHIR as a single standard for the representation of annotated texts. This standardization results in the reuse of existing annotation datasets for different NLP tasks and easy integration with each other. This saves time and cost. The study demonstrated that the performance of an NLP tool is enhanced when it is trained using standardized and aggregated annotation data. In summary, the study succeeded in demonstrating how to use the FHIR standard to improve the compatibility and reusability of various clinical text annotation datasets.

In spite of its contributions, the study has a number of limitations. To begin with, it was found that only a portion of the elements in some FHIR resources could be mapped to the elements in the original dataset. This happened during the process of mapping the original annotation schemas to the FHIR schema. As a result, many elements, that were defined in FHIR, did not have corresponding annotations in existing datasets. FHIR lacks built-in elements for representing some important NLP concepts. The evaluation of performance of the NLP engine, that was trained on the aggregated FHIR



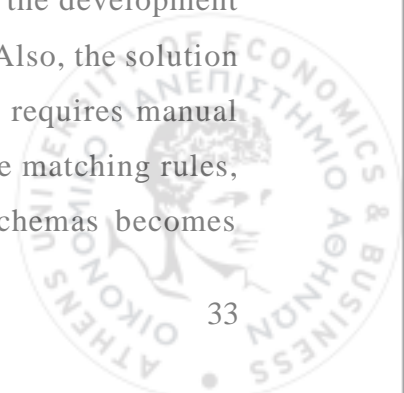
data, focused on extracting specific elements. Last but not least, it is the issue of privacy and confidentiality of health data since it contains sensitive clinical data.

To address these limitations, several steps are followed. The challenges are solved by continuous data integration, development of FHIR extensions, future assessments, collaboration on privacy protection, improvements in the flexibility of schema creation, and the adoption of more formal and automated methods in order to manage transformation and normalization rules.

Additional research that is examined in the one that is carried out by Schneider, Julian Moreno, et al. [29]. The purpose of this study is to present approaches that address compatibility problems in NLP platforms. The study examines both the manual creation of matching rules as well as automating this process using advanced NLP models. The main goal is to achieve semantic compatibility between NLP services, despite the way of marking up information.

Based on this study, several outputs and results are produced. The study presents three different situations where incompatibilities appear between NLP platforms. For each scenario, the study outlines the approaches that are used to deal with these problems. The first scenario includes the development of a common API specification. A common API specification is like a set of rules that defines the way that different software systems communicate each other. Services that meet this specification can be easier combined. The second scenario involves is handling different data formats via a workflow manager. The primary outcome is the use of an intermediate unified format, RDF, to simplify format conversions. The third scenario is addressing compatibility using various semantic schemas. The study also highlights the complexity of compatibility when services use different semantic schemas, although they use the same data format. It is challenging to develop universal matching rules for these schemas since they are not formally documented. Overall, the study shows many approaches to ensure compatibility across NLP platforms, with a focus on the challenge of semantic mapping.

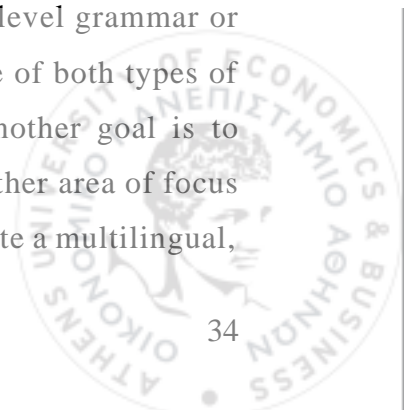
In order to deal with the limitations, the study takes a number of corrective actions. Due to the fact that there is a large number of possible data formats, the development of manual conversions for each possible format pair is not practical. Also, the solution of using RDF as an intermediate unified format is valid, but it still requires manual mapping to and from that format for each new format type. As for the matching rules, creating them for each pair of services with different semantic schemas becomes



unmanageable when the number of services increases. Furthermore, the model is limited in matching more complex semantic patterns. The approach in the first scenario requires all services to use a predefined API specification. This may be a limitation for integrating existing services that do not comply to these specifications, even though the specifications are flexible. While the internal API specification enables combining services, converting responses into requests for services is difficult. In some scenarios where response fields contain arbitrary content, conversion may not be possible. Overall, the study highlights that while steps have been made to improve compatibility, semantic compatibility is still an important challenge with current limitations, especially when it comes to scalability and automation.

The limitations that are mentioned in this study are overcome with many ways. Firstly, the lack of scalability of manual format mapping is resolved with the use of RDF. Instead of requiring conversions for each pair of formats, only conversions to and from the intermediate format are needed. Also, the lack of scalability and generalization of manual semantic mapping is attempted to be solved with automatic semantic mapping. The study introduced preliminary experiments as a possible technique for automatically producing matching rules between various semantic schemas. Moreover, the restriction of the need for a predefined API specification is diminished. This happened due to the fact that some API specifications are designed to be adaptable and cover as many NLP use cases as possible. Last but not least, the unreliability of the automatic message conversion is recognized, particularly for responses that contain annotations or classifications.

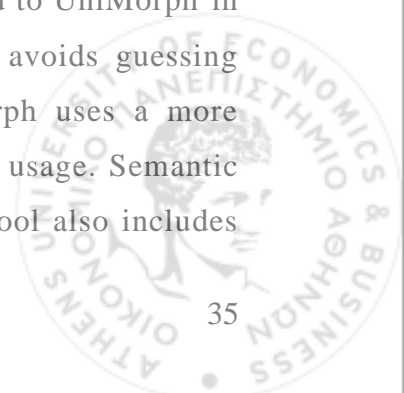
Last but not least, McCarthy, Arya D., et al. developed another significant study [30]. The goal of this research is the development of a mechanism between two multilingual schemes of morphological and syntactic analysis. The specific goal of the study is to create a deterministic mapping. This includes developing a tool that can convert lexico-grammatical descriptions of Universal Dependencies (UDs) into Universal Morphology (UniMorph) format. It should be noted that UD is a system for annotating grammar across many languages, while UniMorph focuses on word-level grammar or how words change their form. This compatibility will enable the use of both types of resources in several tasks such as syntactic structure analysis. Another goal is to identify differences in notes within and between the two works. Another area of focus for this study is enhancing the two annotation systems in order to create a multilingual,



universal catalog of morphological features. In practice, this research focuses on creating a specific mapping between the two different annotation schemas. Also, addressing situations where there are differences in terminology, in the level of detail, or in the existence of specific features in the two schemas is another focus of this study. Last but not least, it needs to be shown that a universal match is not always sufficient due to linguistic characteristics and decisions that are made by the dataset creators. Therefore, this study includes the development of language-specific rules and post-processing steps to enhance the accuracy of the conversion. Overall, this research is an attempt to achieve compatibility between two important resources of morphological analysis. This is achieved by creating a deterministic mapping method between the corresponding annotation schemas.

There are many outputs and results of the research. First of all, a deterministic mapping from UD morphological annotations to the UniMorph schema is developed. This mapping consists of some language-specific modifications for many languages. This research also provides a tool for annotating CoNLL-U files with UniMorph annotations. Overall, a practical and deterministic tool is successfully developed for converting morphological annotations between UD and UniMorph. This enables potential collaboration between these two important resources and highlights points for improvement.

Despite their advantages, the annotation techniques that are employed have limitations. To begin with, while both UD and UniMorph struggle for universality, they are not implemented in practice. Decisions and errors that are made by human annotators, introduce inconsistencies across datasets and languages. Also, many morphosyntactic features are skipped even though they may be accepted by the schema. This results in information loss and difficulty in aligning equivalent annotations across schemas. Moreover, UD includes language-specific features, and UniMorph uses opaque tags, which makes mappings unclear without language-specific knowledge. Both schemas have trouble to represent grammatical patterns that involve multiple words. Words that have many morphosyntactic meanings, can be hard to match. Also, some information from UD is intentionally not mapped to UniMorph in order to avoid unverifiable or misleading features. The mapping avoids guessing values that are not inside the source annotations. Also, UniMorph uses a more linguistically complete schema, while UD is adapted from practical usage. Semantic mismatches make direct translations more complex. The mapping tool also includes



few languages that have both UD and UniMorph resources. Some languages still may show no improvement due to poor annotation alignment or data deficiency. These limitations represent the complexity of balancing annotation schemes and highlight the need for manual post-editing, language-specific rules, and careful treatment of under-specified features.

In response to these limitations, several methods were employed. Due to the fact that the schemas were used inconsistently, the datasets were treated as imperfect, and the focus is on converting the actual annotations they include. The conversion was developed to be robust to inconsistencies. For the annotation gaps, a post-editing phase was developed after an initial mapping. This phase is based on attribute-value pairs. For the language-specific annotations, the authors introduced language-specific conversion rules in order to deal with inconsistencies. As for the information loss during conversion, they made conservative decisions, while avoiding any additions that were not reliable. For differences in structure between schemas, the team developed an initial language-agnostic mapping of many attribute-value pairs, and then iteratively refined this to deal with mismatches. This research also focused on some languages where both UD and UniMorph data were available. For each schema, they developed custom rules that are based on comparisons.



CHAPTER 3

This chapter explains the steps that are followed to test the annotation mapping method that has been developed in this research. It starts by showing the results from the RAA tool and the annotations that have been created through this tool. This is the main source of data that is used in the analysis. Then, it describes how the Python script is used to match and map each research artifact from the RAA tool to the right sentence, as created by the INCEpTION tool. The process also includes ways to handle difficult cases, like when sentences are not exactly the same. To improve the results, the LCS method is used. At the end of the chapter, both qualitative and quantitative results are shown to check the accuracy and reliability of this method.

3.1 Annotation Output from RAA

The research starts with the utilization of the RAA tool. Figure 1 below displays the first sheet of the XLSX file which serves as the output of the RAA tool. The “Publication_Info” sheet includes metadata that is associated with the research article under study. It contains an MD5 hash identifier, which acts as a unique point of reference for document tracking. The sheet also provides the full title of the research paper, along with the names of the authors. Additionally, it contains the abstract and the sections, which is a summary of the document that lists important sections. The section titles are efficiently listed in this part. By organizing textual and bibliographic metadata from research publications, the RAA tool facilitates the analysis and reference of academic documents. Additional sheets in this XLSX file, such as “Research_Artifacts” and “Mentions” offer more detailed explanations of the content of the research paper under study.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Identifiers	MDS: DD00269458702E21CDE58C14464CB473																					
2	Title	THE DETERMINANTS AND EFFECTIVENESS OF INDUSTRIAL POLICY IN CHINA: A STUDY BASED ON FIVE-YEAR PLANS																					
3	Authors	Yiyun WuXiwei ZhuNicolaas Groenewold																					
4	Abstract	Industrial policy is ubiquitous in both developing and developed countries. In China, government intervention in favor of specific industries is believed to play an important role in economic development. The targeting of strategic industries for pre																					
5	Sections	IntroductionBackground: China's Five-Year PlansData and stylized facts(Insert Table 2 around here)(Insert Table 3 around here)(Insert Table 4 around here)(Insert Table 5 around here)(Insert Table 6 around here)Provincial governments' choice of i																					
6																							
7																							
8																							
9																							
10																							
11																							
12																							
13																							
14																							
15																							
16																							
17																							
18																							
19																							
20																							
21																							
22																							
23																							
24																							
25																							

Figure 1: RAA xlsx output file - sheet 1.

The second sheet of an XLSX file, that is called “Research_Artifacts”, is displayed in Figure 2. The sheet includes structured information that is related to a number of research artifacts that are mentioned in the publication for analysis. Each row represents an individual research artifact, which is identified by a unique Artifact ID. Artifacts are grouped into different RA Clusters, based on their function or domain. The “Research Artifact” column also gives the specific names of the identified artifacts.

Additionally, in the dataset, each artifact is classified by type, distinguishing between categories. It further records whether the artifact is owned by the authors of the analyzed publication and whether it has been reused in the research. Additional columns provide metadata such as licensing information, version details, URLs, and citation records. The “Citations” column shows the years in which the artifact was cited as stated in this research paper, while the “Mentions Count” column records the number of times each artifact name appears in the analyzed XLSX file, in the “Mentions” sheet. This sheet is essential for mapping how research artifacts are used, cited, and interconnected within academic literature.

	A	B	C	D	E	F	G	H	I	J	K
1	Artifact ID	RA Cluster	Research Artifact	Type	Owned	Reused	Licenses	Versions	URLs	Citations	Mentions Count
2	A1	software_cluster_5	TFP	software	No	Yes					1
3	A2	software_cluster_3	propensity score matching	software	No	Yes				(2006)(2007)(2008)(2009)(2010)	1
4	A3	software_cluster_4	propensity-score matching	software	No	No				(1983)	1
5	A4	software_cluster_1	PSM	software	No	Yes					1
6	A5	software_cluster_2	PSM-DID	software	No	Yes				(2)(4)	2
7	A6	software_cluster_0	GMM	software	No	Yes				(2)(3)(5)(6)	1
8	A7	software_unnamed	Unnamed_6	software	No	Yes					1
9	A8	software_unnamed	Unnamed_7	software	No	Yes					1
10	A9	software_unnamed	Unnamed_8	software	No	Yes				(2012)(1996)	1
11	A10	software_unnamed	Unnamed_9	software	No	Yes					1
12	A11	software_unnamed	Unnamed_10	software	No	Yes					1
13	A12	dataset_cluster_10	higher	dataset	No	No				(1994)	1
14	A13	dataset_cluster_7	China Industrial Statistics Database	dataset	No	Yes		2015			6
15	A14	dataset_cluster_9	provincial panel data	dataset	Yes	Yes					1
16	A15	dataset_cluster_8	industrial data	dataset	No	Yes					1
17	A16	dataset_cluster_3	CSMAR	dataset	No	Yes					1
18	A17	dataset_cluster_6	Annual Survey of Industrial and Survey	dataset	No	Yes					1
19	A18	dataset_cluster_5	Annual Survey of Industrial Firms	dataset	No	Yes					1
20	A19	dataset_cluster_4	AFSI	dataset	No	Yes				(2012)(1996)	1
21	A20	dataset_cluster_1	PPI	dataset	No	Yes					1
22	A21	dataset_cluster_2	cycle	dataset	No	Yes					1
23	A22	dataset_cluster_0	PSM	dataset	No	Yes				(2)(4)	1
24	A23	dataset_unnamed	Unnamed_3	dataset	No	Yes					1

Figure 2: RAA *xlsx* output file - sheet 2.

Last but not least, Figure 3 shows the “Mentions” sheet from the output file of the RAA tool, which significantly contributes to the analysis that is carried out in this study. This sheet offers detailed information about how different research artifacts are mentioned within academic literature. Each row is assigned a specific research artifact, which is identified by a distinct Mention ID.

The sheet arranges information into multiple important columns. To differentiate research artifacts, the “RA Cluster” column groups them according to their category. The “Research Artifact” column lists the specific names of the referenced artifacts. The “Type” column categorizes each artifact in a specific type, while the “Owned” and “Reused” columns show whether the artifact was developed by the authors of the research and whether it has been used in previous studies.

Furthermore, the sheet includes citations that specify the years that each artifact was referenced as stated in this research paper. The “Section” column indicates the part of the research document or section in which the artifact appears. The “Trigger” column specifies the token that triggered the search of a research artifact in this specific sentence of the research paper. One of the most significant columns, the “Mention” column, includes the sentence from the text under analysis where the corresponding artifact is referenced or mentioned. This sheet plays an important role in the research due to the fact that it allows for a structured analysis of how research artifacts are referenced and used in academic literature. In order to improve annotation mapping methods and guarantee that entities are correctly linked within NLP-based text mining

workflows, the research will make use of this structured data sheet from the XLSX file.

Mention ID	RA Cluster	Research Artifact	Type	Owned	Reused	License	Version	URLs	Citations	Section	Trigger	Mention
1	C132	software_cluster_5	TFP	software	Yes	Yes				(Insert Table 10 around here)	IRN	We first use TFP as a regressor in the probit selection equations. Therefore in the third part of section 5 the propensity score matching (PSM) is used to evaluate whether any beneficial effect lasts beyond the current period of the relevant PVP.7 (2006/2007) Therefore, we employ the propensity score matching (PSM) proposed by Rosenb...
3	C158	software_cluster_3	propensity score matching	software	No	Yes			(2006)(2007)(2008)(2009)(2010)	The effect of industrial policy on output	method	We then use the PSM to screen samples and make sure the treatment group and similar at the baseline.
4	C210	software_cluster_4	propensity score matching	software	No	Yes			(1983)	Policy effects in the short and long terms	method	Having established the effectiveness of policy within the plan's period of operation using the PSM now turns to test the existence of a carry-over effect, i.e., whether there is a persisted influence of post-plan period.
5	C238	software_cluster_1	PSM	software	No	Yes				Policy effects in the short and long terms	method	If the logarithm of industrial output or output change is used as the dependent variable in the PS reported in columns (2) (4) of Table 18, the coefficient of the interaction term is no longer significant irrespective of which matching technique is employed.
6	C241	software_cluster_2	PSM-DID	software	No	Yes				Policy effects in the short and long terms	approach	(2) The simple average of firm-level TFP was used instead of the weighted average to compute Inc in the regressions in Table 11 and Table 12. (3) The GMM method was used to account for heteroscedasticity for regressions (5) and (6) in Table 15.
7	C248	software_cluster_2	PSM-DID	software	Yes	Yes			(2)(4)	Policy effects in the short and long terms	model	For industry i in province i, TFP is calculated as the weighted average of firm-level TFP is approximated using firm-level output as weights.
8	C255	software_cluster_0	GMM	software	No	Yes			(5)(6)(2)(3)	Further Robustness Checks	method	More specifically, the probit model is specified as follows:
9	C130	software_unnamed	Unnamed_6	software	No	Yes				(Insert Table 10 around here)	IRN	The AFSI data are cleaned and processed as suggested by Nie et al. (2012); firm-level TFP is computed using the predicted probability of an industry's being selected using a probit model.
10	C119	software_unnamed	Unnamed_7	software	No	Yes				Provincial governments' choice of industrial policy	model	More specifically, we first estimate the predicted selection probability of an industry using a probit model. However, Harrison (1994), using the same data set, showed that more protected sectors productivity growth.
11	C77	software_unnamed	Unnamed_8	software	No	Yes			(1996)(2012)	Data and stylized facts	method	Data used in this paper include industrial policies extracted from the latest four NFIPs and PVP covering the period 1996 to 2015), as well as industry data for 31 Chinese provinces, autonomous municipalities (hereafter abbreviated as provinces) obtained from CSMAA's China Industrial Sta...
12	C115	software_unnamed	Unnamed_9	software	No	Yes				Provincial governments' choice of industrial policy	model	The China Industrial Statistics Database is a sub-database of CSMAA (China Stock Market Tradin...
13	C25	software_unnamed	Unnamed_10	software	No	Yes				introduction	model	The China Industrial Statistics Database is a sub-database of CSMAA (China Stock Market Tradin...
14	C15	dataset_cluster_10	higher	dataset	No	No			(1994)	introduction	data set	The China Industrial Statistics Database is a sub-database of CSMAA (China Stock Market Tradin...
15	C45	dataset_cluster_7	China Industrial Statistics Database	dataset	No	Yes				Data and stylized facts	database	The China Industrial Statistics Database is a sub-database of CSMAA (China Stock Market Tradin...
16	C52	dataset_cluster_7	China Industrial Statistics Database	dataset	Yes	No				Data and stylized facts	database	The China Industrial Statistics Database is a sub-database of CSMAA (China Stock Market Tradin...
17	C53	dataset_cluster_7	China Industrial Statistics Database	dataset	No	No				Data and stylized facts	database	The China Industrial Statistics Database is a sub-database of CSMAA (China Stock Market Tradin...
18	C54	dataset_cluster_7	China Industrial Statistics Database	dataset	No	No				Data and stylized facts	database	The China Industrial Statistics Database is a sub-database of CSMAA (China Stock Market Tradin...

Figure 3: RAA xlsx output file - sheet 3.

3.2 Output from INCEpTION

Moving on to the INCEpTION part, Figure 4 shows the main dashboard of the INCEpTION annotation tool when a user enters the platform. The project that is shown is titled “Research Artifact Automatic Annotation”. The interface consists of a sidebar on the left with many annotation-related options, including “Annotation”, “Curation”, “Monitoring”, “Agreement”, and “Settings”. These features allow users to annotate text, manage annotation workflows and monitor progress. On the right side of the interface, there is a “Recent Activity” panel that shows a list of previously annotated files. This includes a number of recently used files. Each file has a timestamp next to it that shows when certain actions were completed. At the bottom, the interface shows that the tool is operating on version 31.3, developed by Technische Universität Darmstadt’s Computer Science Department. This dashboard acts as the central point for annotation projects that allows researchers to manage, review, and refine text annotations within structured datasets.

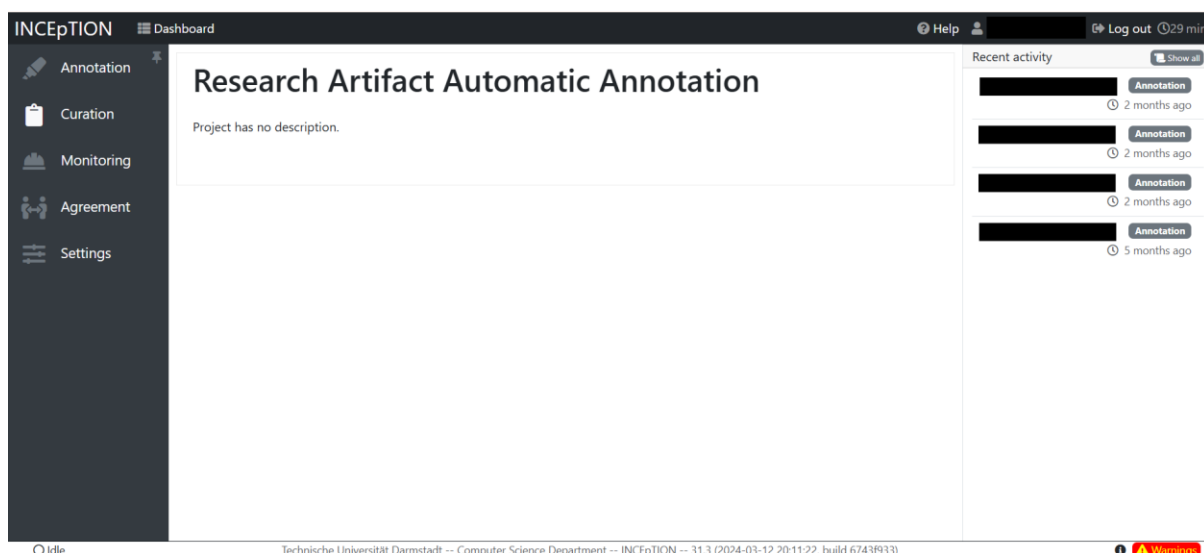


Figure 4: The main dashboard of the INCEpTION annotation tool.

The first step of this research includes converting the research paper under study that has also been analyzed by the RAA tool, from PDF format to CAS format. This is necessary since the annotation mapping is going to be performed from the XLSX file of the RAA tool to the CAS file of the INCEpTION. To begin with, in the INCEpTION annotation tool interface, the “Settings” and then “Documents” from the left sidebar are selected. This step serves as the first stage of the process, which enables importing a PDF file into the system for manual annotations or to convert it into the CAS format for later use. This conversion is required due to the fact that the research paper is processed in the analysis script using its corresponding CAS file format.

After the preceding steps are finished, the interface that is shown in Figure 5 appears. This interface displays the imported documents and their related metadata, plus the file format, size, and creation date. The “Name” column lists the filenames, and the “Format” column provides the format of each document. The “Size” column specifies the file size, while the “Created” column states the date when each file was uploaded into the system. On the top of the interface, there is a file import section where users can select a file, specify its format, and import it into the INCEpTION tool. Additionally, on the right side, users have options to delete or export files.



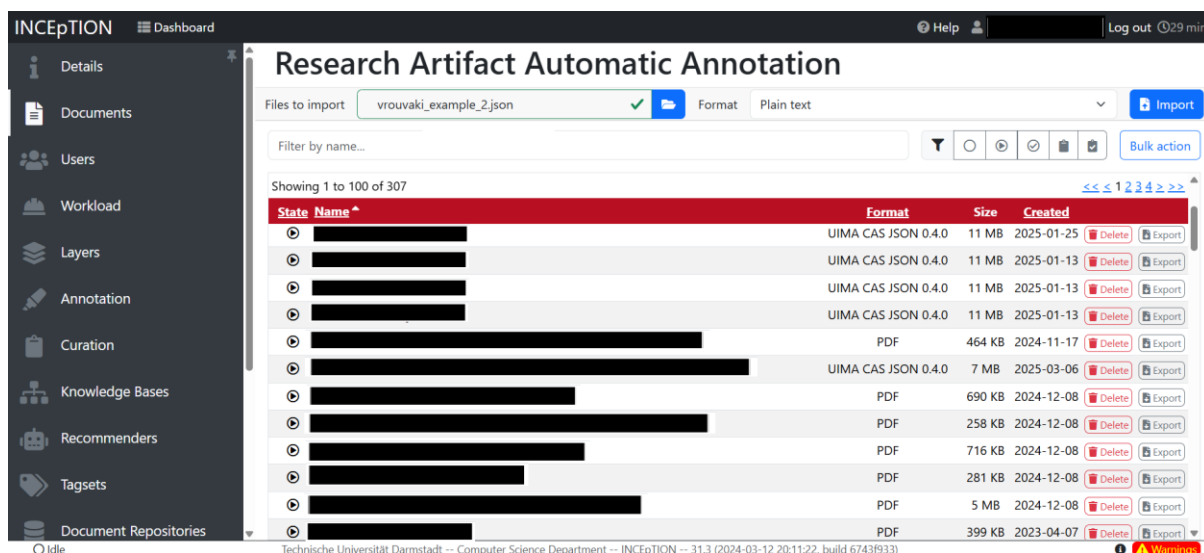


Figure 5: Document import interface in INCEpTION.

After uploading and importing the research paper in PDF format into the INCEpTION platform, this specific research paper is chosen. Figure 6 displays the INCEpTION interface after choosing the “Annotation” option from the left sidebar. The user can select a file for annotation by using the document selection window that appears on the screen. Several icons for sorting and filtering options are included in the interface, along with a search bar that enables document filtering by name. The document list contains multiple format files which indicate that the system supports a variety of file formats for annotation. This screen is an essential part of the annotation workflow, as it allows users to load documents for text processing, annotation, and further curation within the INCEpTION framework.

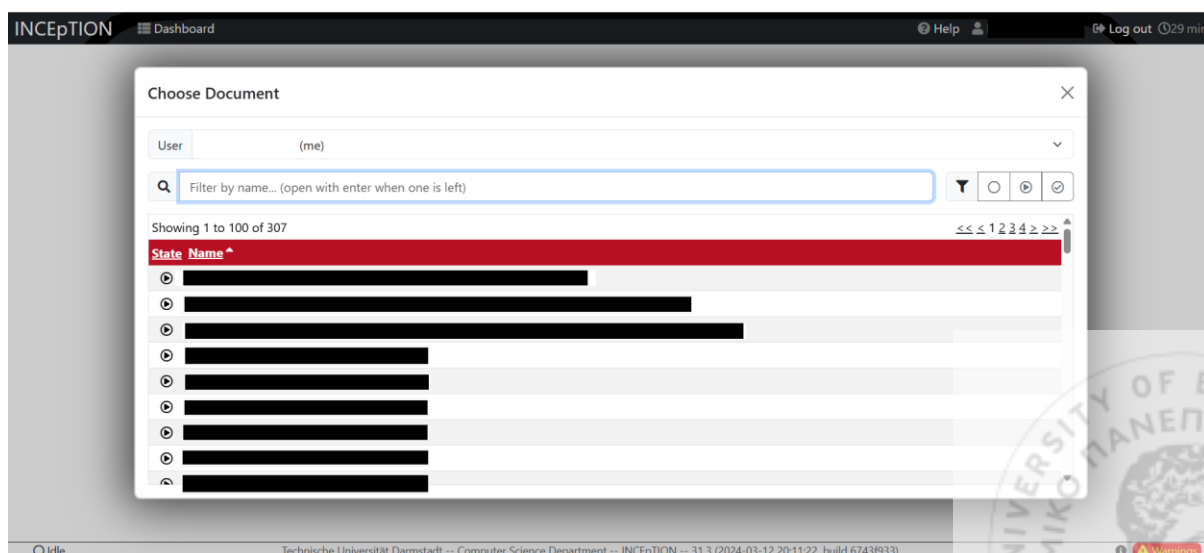


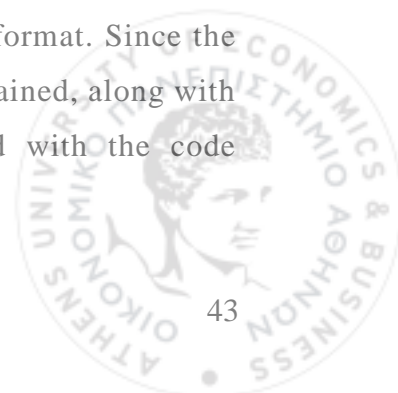
Figure 6: Document selection interface in INCEpTION for annotation.

After the PDF file for further study has been chosen, another interface is shown, that is depicted in Figure 7. This image shows the interface that presents a PDF research paper that is loaded into the system and is prepared for annotation. The entire document file is successfully imported, and its corresponding text is presented. Options for grouping annotations by label and sorting them based on scores or recommendations are available on the left panel. The toolbar at the top offers a number of features. The right panel focuses on the annotation process, where selected text segments can be labeled within predefined categories. The active annotation layer is set to “Research Artifact”, which suggests that the main goal is on identifying and classifying relevant entities that are related to research artifacts. This stage is necessary for getting the document ready for structured annotation and enable researchers to manually label important text segments before exporting them into CAS format. In this research, this step is crucial for downloading the research paper in CAS format, by choosing the “UIMA CAS 0.4.0” option.



Figure 7: INCEpTION annotation interface for manual labelling or exporting the PDF in CAS format.

As a result, the research paper is successfully downloaded in CAS format. Since the XLSX file that includes the analysis from the RAA tool has been obtained, along with the research paper in CAS format, the next step is to proceed with the code implementation phase.

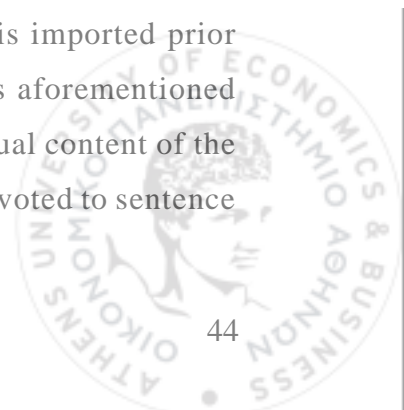


3.3 Code Implementation

Following the preprocessing stages, annotation mapping is carried out using the XLSX file that is produced by the RAA tool and the CAS file that is extracted from the INCEpTION platform. This step ensures the alignment of annotations across various tools, which facilitates structured data representation. The mapping technique is developed within a Jupyter notebook, using the Anaconda Navigator environment [31] to guarantee a productive and structured development process. The primary objective of this Jupyter notebook is to process and map annotations from the XLSX file of the RAA tool to the CAS format of the research paper under study, which can then be loaded into the INCEpTION tool. The notebook is structured into multiple sections that cover multiple aspects of data processing and annotation mapping.

The Jupyter Notebook starts by importing a number of Python libraries and frameworks that are necessary for the mapping process. These libraries ensure a productive and organized workflow as they simplify file handling, text processing, and data management. In order to manage structured datasets, “pandas” is used which enables for efficient manipulation of tabular data. Ensuring alignment across different tools is a crucial aspect of annotation mapping. This is achieved through the use of “pylcs”, an important algorithm for text segment alignment that calculates the LCS between sentence parts. Additionally, UIMA CAS files are processed using “cassis”, allowing for conversion and structured annotation manipulation. By including these libraries, the notebook ensures an efficient annotation mapping workflow for processing textual data, structuring annotation management, and optimizing alignment across different annotation tools. Last but not least, the “copy” provides functions that create deep copies of Python objects.

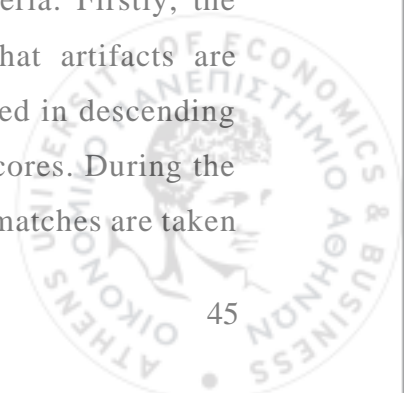
After loading the necessary libraries and frameworks, the CAS file of the research paper under study should be loaded in the Jupyter notebook. The proper Typesystem, which is an XML-based schema that defines annotation structures, is imported prior to the CAS file. The CAS file in JSON format is then read with this aforementioned Typesystem. Following this, its sofa string, which represents the textual content of the research paper, is extracted. After that, a section of the notebook is devoted to sentence



extraction from the CAS file of the research paper. The text of the research paper is divided into sentences, which is used for text segmentation, along with the beginning and ending point of each sentence. This process forms the CAS dataset, as it is referred to below.

The script then reads the XLSX file, which is the output of the RAA tool, and extracts information from the “Mentions” sheet. After loading the data, the script performs a cleansing process, in which it eliminates particular XML-like tags from the “Mention” column. It should be noted that the “Mention” column includes the sentences from the research paper that include the research artifact for annotation. These tags are used to identify entities in text, but they are not required for additional processing. Instead of removing them entirely, the script replaces them with spaces to ensure proper readability. After cleaning the “Mention” column, the script further enhances text formatting by eliminating additional spaces. To ensure consistency, consecutive spaces, which often appear during text cleaning operations, are replaced with a single space.

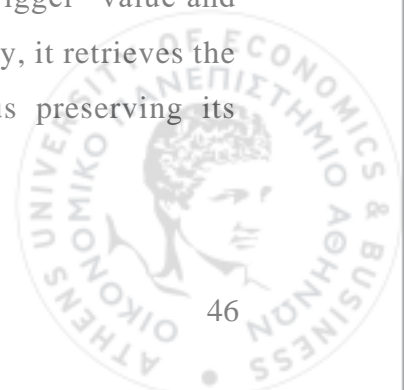
When the preprocessing state is finished, the execution phase begins. To begin with, for each pair of research artifacts and corresponding sentences in the XLSX file, in the “Mention” column, the script searches the CAS dataset for matching sentences. The filtering process initially keeps only those sentences that contain the research artifact. The script determines textual similarity for every pair of sentences from the CAS dataset and XLSX file, using the LCS algorithm. This procedure involves calculating the LCS length between the “Mention” sentence and the CAS sentence, along with a similarity percentage. The ratio of the LCS length to the maximum length of the two texts, multiplied by 100, results in the similarity percentage. This metric offers a numerical base for evaluating the level of alignment by measuring the proportion of the “Mention” sentence that appears in the matched CAS sentence. Following the similarity calculations, each research artifact along with the “Mention” text, CAS sentence and the corresponding calculations are documented. The execution continues by keeping only the most relevant match for each research artifact. To facilitate selection, the similarity results are sorted using two criteria. Firstly, the research artifact index is sorted in ascending order to ensure that artifacts are processed sequentially, and second, the similarity percentage is sorted in descending order to prioritize the pair of sentences with the highest similarity scores. During the selection process, this sorting method ensures that the most relevant matches are taken



into account first. Following data sorting, the script groups the similarity results by research artifact index and handles each artifact separately. The pair of sentences with the highest similarity score is then chosen. One CAS sentence is selected as the most pertinent match only if it contains the research artifact. By using this technique, the script ensures that research artifacts are mapped to their original contextual references in the CAS dataset.

For those research artifacts that are not found in any sentence in the CAS dataset, the script improves the text similarity matching process by implementing a number of new comparisons. In this method, text is normalized by eliminating spaces and mentions and CAS sentences are compared using LCS similarity analysis again. The script then ensures text normalization by eliminating spaces from CAS sentences, mentions, and research artifacts. By removing inconsistencies that are brought on by different spacing, this step increases the precision of text matching. Having normalized the text, the script then filters the CAS dataset and finds sentences that contain the relevant research artifact. This enables the extraction of relevant CAS sentences that include the research artifact. After identifying the relevant CAS sentences, the script calculates the LCS similarity between the XLSX mention and each candidate CAS sentence and derives a similarity percentage. This percentage determines how closely the XLSX mention matches the CAS sentence. The sentence with the highest similarity score is then kept.

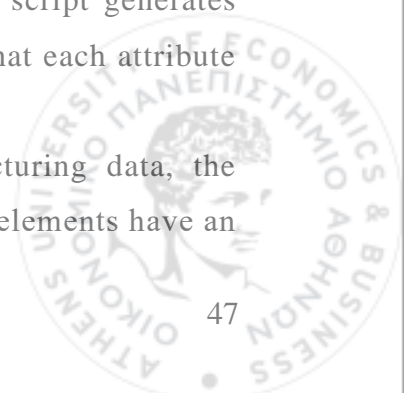
For research artifacts that are indirectly referenced in their corresponding XLSX mention, the script implements an alternative matching mechanism using the “Trigger” column from the XLSX file. Rather than relying only on the research artifact, it looks for the “Trigger” value in the CAS dataset to spot relevant sentences. This methodology is also valid for the research artifacts that are implied, rather than directly stated. The LCS similarity calculation is then used again to locate the most similar sentence, ensuring that the mentions with indirect artifact references still have a corresponding CAS sentence. The script further refines this approach by choosing the best match for each XLSX and CAS sentences. In order to extract the highest-ranked sentence, it groups the matches according to their ID and “Trigger” value and then iterates over the grouped results. In order to maintain consistency, it retrieves the original CAS sentence that corresponds to each best match, thus preserving its unprocessed textual form.



In order to further refine the text similarity matching process, another attempt is performed to find triggers in CAS sentences after converting them to lowercase. This extra step is used to identify matches that might have been missed in the previous matching process because of case sensitivity concerns. The first step in the process is filtering to only include the cases that were missed in earlier iterations of similarity matching. The “Trigger” and “Mention” columns along with the CAS sentences are changed to lowercase, in order to make sure that any differences in capitalization do not interfere with the matching process. After normalizing the text, the script looks through the CAS dataset to find sentences that contain the lowercase “Trigger” value. The sentences that match are processed further. For the sentences that contain the “Trigger” value, the script continues with LCS similarity analysis to find the best match. The LCS length is used to calculate the similarity percentage. Following the identification of the most relevant sentences, the script sorts the matches by similarity percentage in descending order to prioritize the highest similarity sentence. After that, it iteratively goes over the sorted data, making sure that every mention is matched with the most relevant sentence. Only the best match for each research artifact is kept, increasing annotation accuracy.

The next section of the script is designed to create annotations for research artifacts and their corresponding metadata into the CAS object. The procedure starts with getting the necessary annotation types from the CAS Typesystem. These annotation types are the components that are needed to label the research artifacts or “Trigger” values. After loading the annotation types, the script extracts the starting and ending positions of each research artifact and their corresponding metadata. By utilizing this information, a Research Artifact annotation is created for each artifact. Each annotation has important attributes such as its class, type, and span positions, most of which are found in the XLSX file of the RAA tool or have been calculated inside the script. The annotation is added to the CAS object and the artifact information is integrated into the CAS file. To further enrich the annotations, the script utilizes additional metadata attributes, including licenses, version, and URLs that are also found inside the XLX file. For each of these metadata fields, the script generates individual Research Artifact Metadata annotations and guarantees that each attribute is correctly mapped to its corresponding span position.

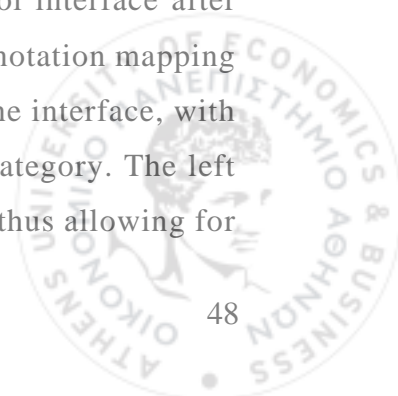
Two important annotation types play an important role in structuring data, the “ResearchArtifact”, and “ResearchArtifactMetadata”. Each of these elements have an



important role in organizing research artifacts and their associated metadata in a structured manner. The “ResearchArtifact” annotation is responsible for creating annotations that are related to research artifacts or triggers within the text. As previously mentioned, a research artifact can be any referenced entity such as a dataset, software, publication, or others. This annotation contains attributes that classify the artifact and defines whether it is owned, reused, or referenced. Furthermore, the type of artifact, such as a publication, software, or dataset, is specified. The annotation additionally provides start and end positions that specify the precise textual span in which the artifact appears in order to guarantee accurate identification. These features give a representation of research artifacts and allow them to be found and classified within the CAS file. In addition to defining the research artifact, further metadata needs to be annotated for each artifact for a better representation. The “ResearchArtifactMetadata” annotation is used for this purpose by keeping additional information about the artifact, such as licensing details, version numbers, and relevant URLs. Each metadata entry indicates the start and end positions that specify the section of text where the metadata is referred.

These two annotation types cooperate to form a hierarchy that arranges research artifacts and their metadata in an efficient way. The “ResearchArtifact” annotation defines the main entity, while the “ResearchArtifactMetadata” annotation stores additional information. This approach allows for efficient data representation and ensures that research artifacts along with their metadata are properly defined within the CAS file. By using this approach, the annotation system offers a structured depiction of research artifacts, enabling more accurate analysis in the fields of knowledge extraction and processing scientific literature.

The final CAS file with annotations is then saved in JSON format. This JSON file serves as the final output of the script, including all research artifact annotations as well as their metadata and positions. To confirm the validity of the process described above, the extracted CAS file in JSON format is uploaded again into the INCEpTION platform, by following the process described in Figures 4-7. The resulting annotated file is shown in Figure 8. This Figure displays the INCEpTION tool interface after uploading and opening the CAS file, that is generated through the annotation mapping process in the script. The annotated research paper is displayed on the interface, with various research artifacts which are categorized according to their category. The left panel lists the annotated research artifacts that are grouped by label, thus allowing for



an organized view of the annotations. The main document in the center shows the annotated text with highlighted parts that correspond to research artifacts, each one colored based on their category. On the right panel, the annotation properties are displayed, showing some important details. This step validates the reliability of the annotation process by visualizing the annotations and their alignment within the text. The approach guarantees that the mappings that are produced by the code are accurately visualized in the INCEpTION tool.

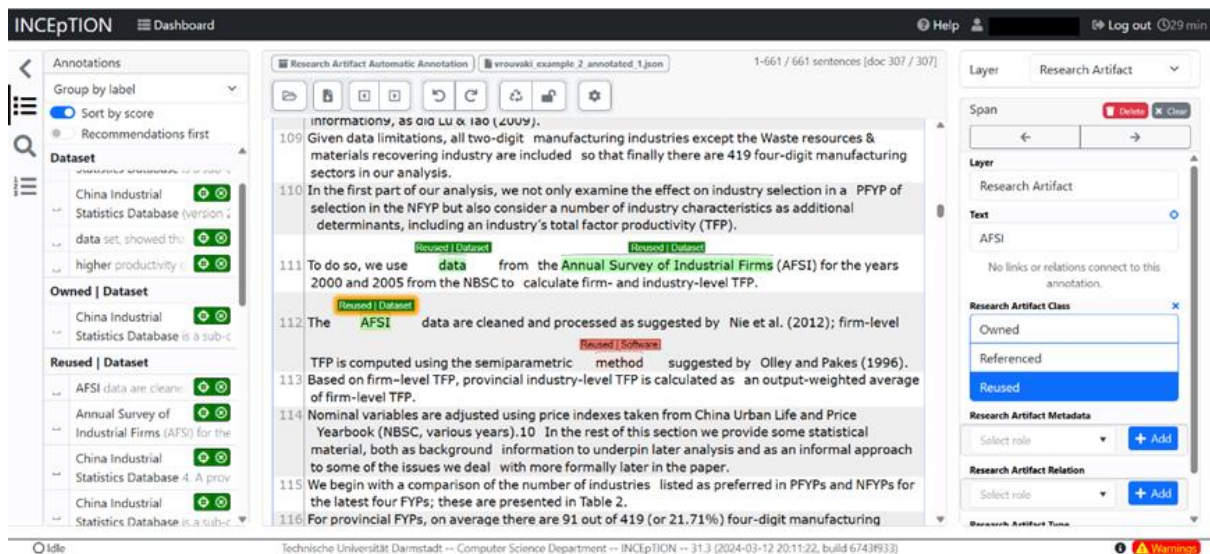
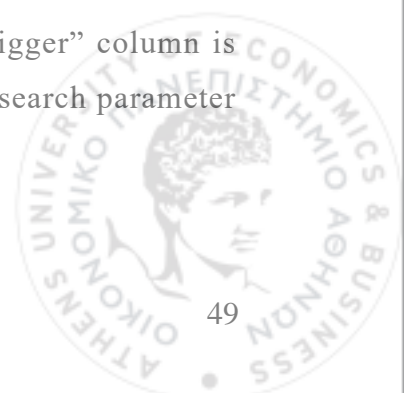


Figure 8: Annotated research paper in INCEpTION.

3.4 Annotation Mapping Challenges

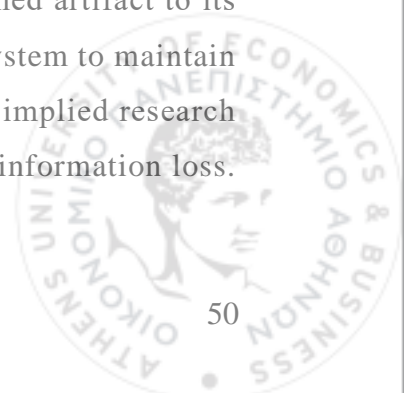
When handling edge cases such as partial matches, overlapping sentences, and encoding mismatches in NLP annotation mapping, a number of situations can occur that require careful thought. To begin with, partial matches can happen when a research artifact is not identical to its reference in the Mention or CAS text but still shares some similarities. There are indeed some cases where the name of the research artifact is not identical to its reference in the corresponding mention in the XLSX file or CAS dataset. In this case, the corresponding value from the “Trigger” column is used from the RAA analysis, and this “Trigger” value is used as the search parameter within the respective mention or CAS dataset.



Another difficulty arises when research artifacts are referenced in several sentences. In some cases, an artifact is introduced in one sentence but expanded upon the next or another sentence. For this reason, it is required for the annotation tool to decide how to align and distribute the annotation across the text. A research artifact may appear multiple times within consecutive or different sentences, which makes it unclear whether both instances should be annotated or just one. This is not applicable in this research, as the annotations are generated on specific mentions, as specified by the RAA tool.

Encoding mismatches are another significant issue, particularly when different NLP tools employ a variety of text encoding standards. This may lead to misaligned text or inconsistent annotations. Special character distortions are usual, particularly when working with non-ASCII characters, such as scientific symbols. Mismatches in Unicode normalization forms may also result in variations. This results in differences in storing or processing text. Invisible characters, such as extra whitespace, line breaks, or zero-width spaces, further ruin string matching and annotation alignment. Additionally, encoding incompatibilities between platforms may cause research papers to display annotations correctly in one tool but may appear corrupted when processed by another tool using different encoding. In these situations, the corresponding value from the “Trigger” column from the RAA analysis is used, and this “Trigger” value is utilized as the search parameter within the respective Mention or CAS sentences.

In many cases, research artifacts are not explicitly mentioned by their original name within a mention in the XLSX file or in the CAS dataset, but instead they are implied through descriptive references. This poses an issue for annotation processes, as direct name extraction is not feasible in these cases. To address this problem, a methodical approach is used where such artifacts are assigned another name in the format “Unnamed_{number}” as shown in Figure 3 in the XLSX file. In these cases, the “Trigger” column plays a crucial role in the annotation process. Since the original name of the research artifact is not present in the sentence, the corresponding “Trigger” value is used as the reference. The trigger links the unnamed artifact to its correct indirect reference in the text, which enables the annotation system to maintain accuracy. By relying on the “Trigger” value, the model ensures that implied research artifacts are correctly mapped within the document, thus preventing information loss.

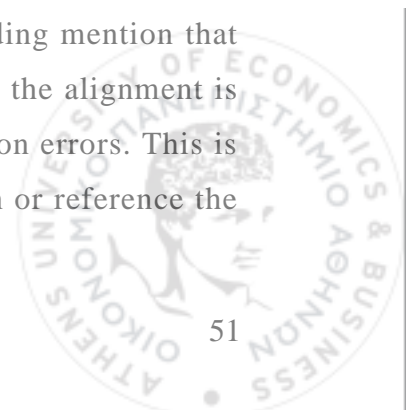


This approach strengthens the robustness of annotation mapping, by guaranteeing that research artifacts, whether explicitly stated or inferred, are included into the analysis. Last but not least, a major challenge in this research study is linking the Research Artifacts with their corresponding metadata. While the script correctly adds both the main Research Artifact annotations and their associated metadata annotations, it does not create a direct link between them. In other words, there is no method that was found that connects a “ResearchArtifact” annotation with its respective “ResearchArtifactMetadata”. As a result, although all important information is stored in the CAS file, it remains isolated from one another. This means that there is a lack of structural connection that would indicate which metadata belongs to which artifact. This limitation means that applications or users that study the CAS file would have to infer these relationships based solely on context, rather than through explicit annotation linkage.

To further address these issues, preprocessing techniques such as text normalization and token-based alignment methods are used. To reduce errors that are brought on by partial matches, the LCS algorithm is also used to find the text segments that match the most. In cases of encoding mismatches, text preprocessing methods such as whitespace correction further help keep consistency across NLP tools. HITL validation also improves the annotation process, by ensuring that ambiguous cases are examined to increase the precision of research artifact extraction and structured annotation integration. Last but not least, the issue concerning the links between the research artifact annotation and their corresponding metadata annotation still remains unresolved.

3.5 Evaluation Metrics

To verify the accuracy of sentence mapping, a manual validation procedure is carried out to verify the correctness of the mapping process. Each mapped sentence that comes from the CAS file is examined and compared against its corresponding mention that contains the research artifact from the XLSX file, to verify whether the alignment is accurate. The goal of this manual procedure is to minimize annotation errors. This is accomplished by verifying that the extracted CAS sentences contain or reference the



intended research artifact and by examining if the selected CAS sentence is identical or part of the corresponding mention. This is achieved by comparing this sentence from the XLSX file with the corresponding sentence match from the CAS file.

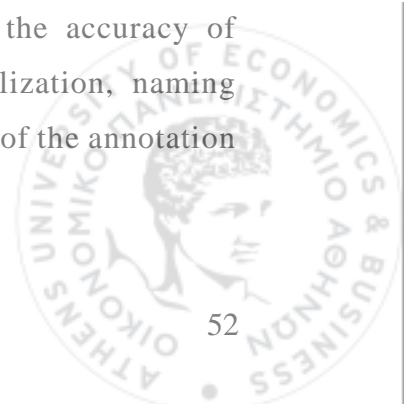
By reviewing each mapping pair, inconsistencies and mismatches are identified and fixed when needed. This validation procedure is crucial in order to evaluate the reliability of the automated mapping approach and determine the efficiency of these techniques. The manual review guarantees that the methodology appropriately captures and connects specific research artifact annotations to their respective CAS sentences and provides high confidence in the results.

3.6 Qualitative Analysis

During the qualitative analysis, it became clear that not every research artifact could be directly identified by its exact name in the corresponding mentions, as it is obvious in the RAA tool. When doing direct name search in the text using the RAA tool, there are inconsistencies because some research artifacts are written in lowercase letters. To address this issue, the algorithm is modified to involve a case-insensitive search, in order to ensure that research artifacts could still be identified even if they did not follow standard capitalization.

Additionally, in certain cases, the name of the research artifact in the “Mentions” in the XLSX file is not identical to the research artifact name, leading to further challenges in direct mapping. In order to overcome this inconsistency, the “Trigger” value is used as an alternative reference for annotation. This allows the algorithm to have a more flexible matching approach when exact name alignment is not possible. Another challenge arose when certain research artifacts are only implied in the sentence rather than explicitly stated by name. In these cases, since no direct reference to the research artifact is present in the mention, the “Trigger” value is again utilized to ensure that the correct context is captured within the annotation process.

These adjustments in the mapping methodology helped increase the accuracy of research artifact identification, ensuring that variations in capitalization, naming discrepancies, and implicit mentions did not reduce the effectiveness of the annotation process.



3.7 Quantitative Analysis

The accuracy of the mapping process can be assessed using performance metrics. These metrics include values such as precision, which calculates the proportion of correctly mapped sentences, and recall, which checks the number of actual research artifacts that are successfully found in sentences. The F1-score is another crucial metric that provides an overall assessment of accuracy by balancing precision and recall. In this research, no performance metrics are taken into account.

Error analysis is also essential in understanding the limitations of the sentence mapping process, in situations where several occurrences of a research artifact appear within a single mention. One significant challenge that was met is when a research artifact appears more than once within a single CAS sentence. In these situations, the algorithm must determine which occurrence to annotate, but because there is no clear differentiation, it marks only the first instance by default. This problem occurs because the script processes CAS sentences with a sequential approach. After identifying the first occurrence of the research artifact in the sentence, it proceeds with annotation without checking whether additional occurrences exist within the same sentence. As a result, if the second or third instance of the artifact is the more relevant one, the mapping might result in errors. This could be problematic when multiple mentions of the same research artifact serve different roles in the sentence.

To address this problem, a more sophisticated annotation technique could be developed, such as redefining the matching process to take sentence structure or semantic context into account. Instead of simply annotating the first occurrence, an improved approach might involve determining which instance offers the most meaningful reference or ensuring that all occurrences are recorded when required. By tackling these issues, the precision and reliability of research artifact annotations could be greatly improved.



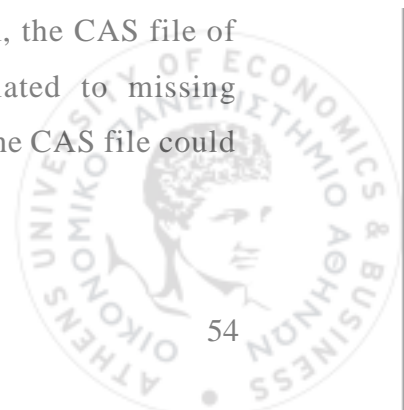
CHAPTER 4

4.1 CAS Typesystem Limitations

One of the main challenges that were noticed during this research is the difficulty in creating the annotations within the CAS file. This issue is caused due to the lack of some annotation types within the CAS Typesystem, specifically “ResearchArtifact” and “ResearchArtifactMetadata”. Since these annotation types are essential for the study, their absence meant that any attempt to create annotations within the CAS file resulted in errors in the annotation process.

As it was previously mentioned, the CAS Typesystem is the schema that specifies which annotation types can exist within a CAS file. Without these predefined structures, the system is unable to identify or support the required annotations. This severely impacted the workflow. Due to the significance of these annotations in mapping research artifacts, an efficient solution was required to incorporate them into the CAS file and enable easy processing.

To overcome this restriction, a two-step method was developed which includes manually adding the missing annotation types into the CAS Typesystem. The first step is to utilize the INCEpTION tool to manually create the required annotation types. More specifically, a random PDF document is uploaded into INCEpTION, where new random annotations are manually created. This step ensures that the Typesystem of this annotated CAS file would include these necessary annotation types, enabling them to be used for further processing. The following step involves the extraction of the Typesystem of this file from the INCEpTION tool and its incorporation into the script. This Typesystem is downloaded from the INCEpTION tool as an XML file and is then loaded at the beginning of the script. Following this, when reading the CAS file of the research paper under study using this Typesystem in the script, the required annotation types are present for further use. By using this modified Typesystem, the CAS file of the research document can be processed without any errors related to missing annotations. Since the necessary annotation types are now included, the CAS file could



be manipulated without any problems and allow for efficient annotation of research artifacts.

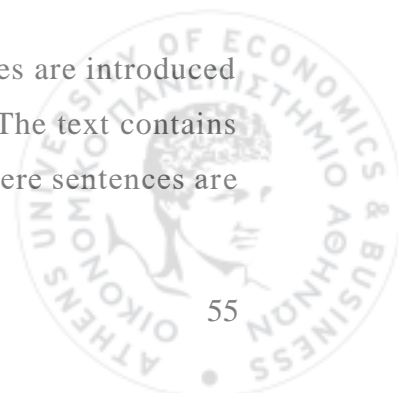
There are several steps in the script that define this procedure. First of all, the script loads the extracted Typesystem XML file and then applies it while reading the CAS file of the research paper. This ensures that the schema that is used in the processing phase contains the required annotations. The updated Typesystem allows the system to retrieve the necessary annotation types without any error, thus enabling it to process and analyze research artifacts within the CAS file efficiently.

This method offers various insights into managing CAS Typesystems efficiently. Having extensive understanding of the structure of the CAS Typesystem is important because it ensures an easy annotation workflow. INCEpTION is essential in resolving this issue as it introduces a way to define and manage custom annotation types. The ability to manually define missing annotation types before processing the CAS file increases workflow efficiency and prevents errors from interfering with the analysis. By putting this solution into practice, it is feasible to incorporate the required annotation types into the CAS Typesystem. This makes it possible to annotate research artifacts efficiently within the document and ensure that the research could proceed without any technical obstacles that are related to annotation types. The approach to addressing Typesystem issues not only leads to successful annotation mapping but also contributes to the overall efficiency of the research process.

4.2 Challenges in Handling Files as JSON

Another challenge that was present during this research is handling CAS files within the Python script. When the research paper is downloaded as a CAS file from the INCEpTION tool, it is saved in JSON format. During the testing phase, the script treated the CAS file as a JSON file and no CAS-related processing methods were used. This method presented serious issues that impacted the accuracy of the generated annotations.

When reading the CAS file as a JSON file, unexpected inconsistencies are introduced in the text or the sofa string as it is well-known in the CAS theory. The text contains abnormal symbols such as ‘\n\n\n’ or ‘\n’, which appear in places where sentences are



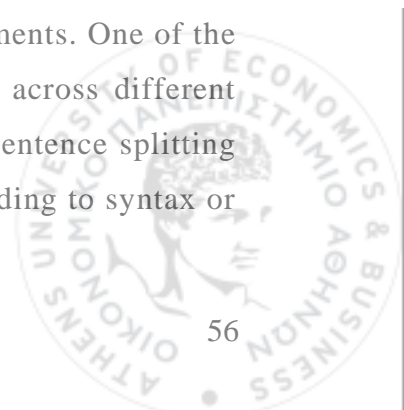
divided across multiple lines in the original document. These symbols are traces of the initial formatting of the text, where a sentence that extended to the next line is interrupted by line breaks. These symbols also appeared when manipulating the CAS file with pandas data frames. In an effort to clean the text, these symbols vanished and produced a cleaner version of the text for further processing.

Although a more readable text output was obtained after text cleansing, there was an issue with the annotations. Since annotations in CAS files use particular character positions to identify research artifacts inside the text, any change in the structure of the text has an impact on the alignment of these annotations. The removal of the symbols for line breaking changes the character positions of words and phrases, which means that the locations of research artifacts no longer correspond to their original positions in the INCEpTION CAS file. As a result, when trying to annotate research artifacts, the assigned positions were incorrect, which led to misaligned annotations and to marking wrong words in the text.

This issue leads to the realization that handling the CAS file purely as a JSON file destroys the original structure. This makes it impossible to maintain the same or the correct character position of the research artifacts. Improper handling of text formatting of the CAS structure ruins the integrity of the annotation process. This verifies the importance of correctly handling CAS files using the appropriate CAS-related methods rather than handling them as plain JSON files. To fix this, the approach was modified to ensure that CAS files are loaded and processed using CAS-related methods that keep character positions correctly. This adjustment allows for accurate mapping of research artifacts and prevents misalignment issues. As a result, the tokens remain consistent with the original text structure.

4.3 Unresolved Issues and Future Work

Despite the advancements that have been made in annotation mapping, a number of unresolved problems and challenges still remain for future improvements. One of the limitations in this research is the difficulty in sentence alignment across different tools. Despite efforts to properly align annotations, differences in sentence splitting methods introduce inconsistencies. Some tools split sentences according to syntax or

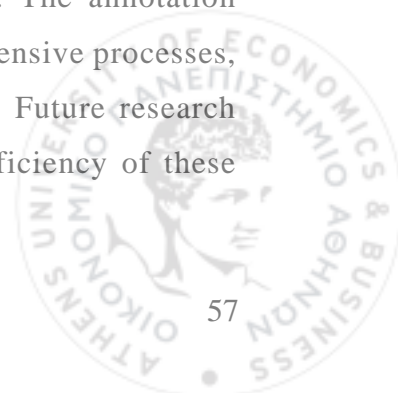


predefined rules, which leads to challenges in guaranteeing perfect mapping. While techniques such as tokenization, standardization and LCS helped to solve these inconsistencies, there is still a need for better alignment mechanisms that can adapt to variations across different NLP frameworks.

Another major obstacle is handling research artifacts that are implied rather than explicitly stated. Some artifacts are referenced indirectly inside the text, which can be challenging to infer their exact reference. In these situations, the “Trigger” value is used in the script as a reference for annotation. However, this method is not completely correct, as it does not involve full understanding of the content. A possible place for improvement is the integration or upgrade of new or existing ML models that can infer implied research artifacts from context instead of relying only on explicit references. The problem of overlapping annotations and partial matches further complicates the issue in annotation accuracy. In cases where a research artifact occurs many times within its corresponding CAS sentence, the script selects the first instance by default. This method ignores cases where a latter occurrence is more relevant. Future improvements should investigate more advanced techniques that take into account the surrounding context, entity importance, and frequency of occurrence to make better decisions.

Encoding mismatches and formatting inconsistencies continue to be another obstacle in achieving successful integration between annotation tools. As seen in the early phases of this research, treating CAS files as JSON files produces unexpected formatting artifacts, such as unnecessary line breaks and special characters or symbols. Even though these problems are eventually resolved by ensuring that CAS files were handled through CAS-related methods, there is still a possibility of text corruption when transferring annotations between different formats. Future studies should concentrate on developing more reliable preprocessing methodologies that maintain text structure while maintaining compatibility across tools.

Scalability is another element that needs further exploration. The current study was conducted using a limited dataset. Although the method proved efficient within this scope, it is unclear how well it would work with larger datasets. The annotation mapping technique in this study does not include computationally intensive processes, but it may become impractical when applied to extensive datasets. Future research should investigate optimization methodologies that increase the efficiency of these



processes, that may include parallel processing or more advanced ML annotation mapping.

Furthermore, linking the Research Artifact annotations to their corresponding Research Artifact Metadata annotations is of utmost importance. The annotation mapping technique in this study does not include developing relations between research artifacts and their corresponding metadata. Future studies should focus on upgrading the script to include these relations in the annotation mapping process, thus preventing information loss.

Last but not least, HITL validation is useful for guaranteeing annotation accuracy, but adds subjectivity and possible inconsistencies. Different annotators may interpret and annotate the same research artifact differently, which can result in variations. Future studies should investigate more automated quality control mechanisms that lower the need for human intervention while preserving high annotation precision.

Resolving these issues can be crucial for improving annotation mapping accuracy, guaranteeing better integration between NLP tools, and refining the methodology for additional applications. To build a more easy and reliable annotation framework, future studies should concentrate on improving context-related research artifact identification, optimizing sentence alignment methods, increasing annotation scalability, and further standardizing annotation procedures.



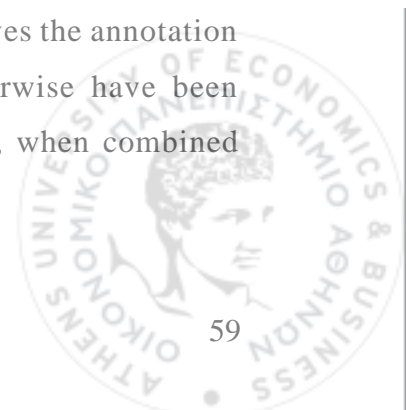
CHAPTER 5

5.1 Analysis of Research Findings

The results of the analysis in this research offer important insights about annotation mapping techniques, particularly when it comes to integrating different tools. In this study, the integrated tools are the INCEpTION and the RAA tools. One of the findings of this research is that structured annotation frameworks, such as CAS files, improve the accuracy and consistency of annotation mapping because they provide a way to organize extracted information across different NLP tools. This study demonstrates that structured representations help in aligning entities and relationships more accurately across different NLP tools. By organizing information, they minimize errors in annotation placement.

A major outcome of the study is the need for handling inconsistencies that arise due to variations in sentence segmentation, tokenization, and encoding schemes among various NLP tools. These inconsistencies cause misalignment in research artifact annotation, thus increasing the need for a reliable preprocessing method to standardize text formats. Encoding normalization techniques are used to address errors that are caused by differences in text representations and ensure that annotations remain consistent even when moving between different processing environments. The study also shows that without proper text standardization, artifacts that are extracted from one system would often fail to align correctly with those from another, which leads to inaccuracies in the annotation process.

Additionally, the study presents automated mapping methods that are designed to reduce the need for manual intervention. By using LCS algorithms and matching techniques, the system could align more efficiently, which makes the annotation process easier and less demanding for human annotators. The use of triggers to map research artifacts that are implied rather than explicitly stated, improves the annotation process, thus making it possible to find artifacts that might otherwise have been overlooked. The study shows that AI-driven annotation techniques, when combined



with structured mapping frameworks, significantly improve the reliability of NLP workflows.

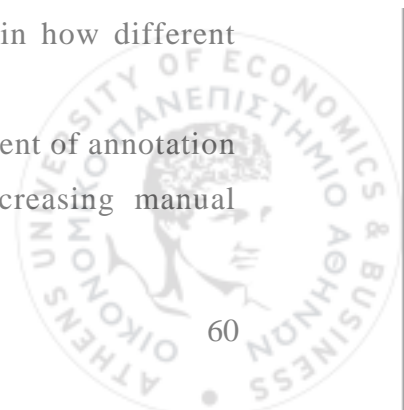
The experiments also point out several challenges that need to be fixed. One major challenge is the difficulty in ensuring compatibility across different NLP frameworks. Despite the implementation of encoding normalization and sentence segmentation adjustments, tools such as INCEpTION and RAA tool still processes text differently, which results in mismatches in annotation placement. These inconsistencies usually need manual correction, which introduce additional time constraints while improving accuracy. Furthermore, cases where research artifacts appear many times within a CAS sentence poses another challenge, as the current system annotates the first occurrence without considering contextual relevance.

Nevertheless, this research makes significant progress in enhancing annotation mapping methodologies with the integration of structured annotation frameworks, automation techniques, and HITL validation. The findings point out the need for refinement in NLP annotation tools to find challenges such as sentence segmentation inconsistencies, tokenization mismatches, and multi-platform compatibility. Despite these challenges, this research efficiently shows that a combination of structured annotation frameworks and mapping techniques leads to more efficient NLP workflows, especially in fields that need precise text analysis and structured data representation.

5.2 Contributions to NLP Annotation

This study contributes to scientific research with its approach to annotation mapping and its reusability across many annotation tools. By creating an annotation mapping methodology, this research improves the accuracy of aligning annotations from various NLP tools. One of the most notable contributions is the ability to integrate the RAA tool with the INCEpTION tool. The methodology that is developed makes sure that annotations stay accurate and aligned despite the differences in how different tools encode and process text.

One of the main advantages of this approach is its efficient management of annotation workflows. By automating the alignment of annotations and decreasing manual



intervention, this study greatly improves the speed and reliability of annotation mapping. By using the LCS method, research artifacts and sentences of both CAS and XLSX files can be precisely matched, ensuring that entities are mapped correctly. Additionally, some annotations are based on the “Trigger” column of the XLSX file, which offers a solution for cases where research artifacts are implied rather than explicitly mentioned in the text, thus refining the annotation accuracy.

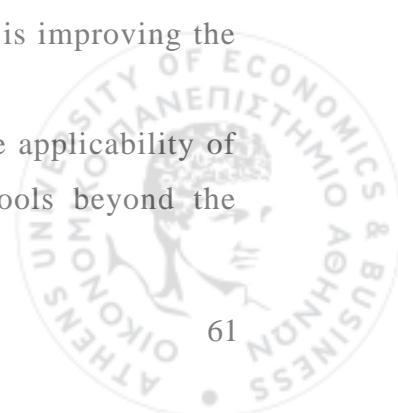
Another significant contribution is the adaptability of this methodology across many platforms. The study demonstrates how the methodology is not limited to specific tools like the RAA or INCEpTION tools but can be extended to other NLP annotation frameworks. By designing a mapping system that is generalized rather than being specific for one tool, the approach offers a solution that can be adapted to many annotation technologies. This guarantees that future NLP tools can integrate easily into the workflow without the requirement of important modifications. The research also highlights compatibility by ensuring that annotations can be transferred between tools without losing information or producing structural inconsistencies.

Overall, this study makes an impact on scientific research with the introduction of a more efficient and reusable approach to annotation mapping. By solving the problems concerning the annotation alignment and compatibility, this research helps in creating more automated workflows for NLP applications. Future work can build upon this methodology and integrate ML-based approaches for extra refinement, in order to ensure greater adaptability and accuracy in annotation mapping.

5.3 Recommendations for Future Research

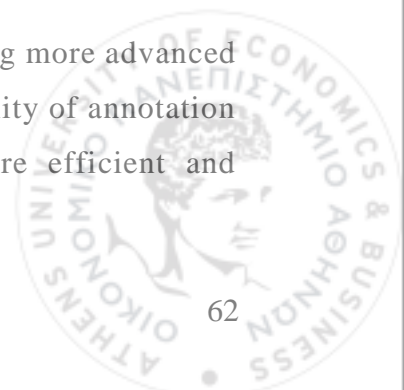
This study shows the usefulness of structured annotation mapping between the INCEpTION and the RAA tools. Nonetheless, further improvements and expansions can improve the accuracy and applicability of the proposed methodology. Future work may focus on two main areas. The first area is expanding the approach beyond the integration of the INCEpTION and RAA tools, and the second area is improving the efficiency by exploring alternative algorithms.

One of the main directions for future research is the increase of the applicability of the annotation mapping methodology to more NLP annotation tools beyond the



INCEpTION and RAA tools. Although this research is based on these tools, the integration of the methodology with other annotation platforms could improve compatibility in annotation workflows. A wide variety of NLP frameworks employ different sentence segmentation or tokenization methods, and metadata structures. This usually leads to inconsistencies in environments that utilize many tools. The expansion of this methodology to adapt to more platforms and to different file formats will enable a more flexible annotation mapping approach. This will make it applicable to a greater number of research fields and industry applications. Moreover, this extension will help to standardize annotation workflows across different tools and improve collaboration between researchers that utilize different annotation platforms. Another aspect of future work focuses on the optimization of the efficiency by investigating alternative algorithms for annotation mapping and alignment. The current study uses the LCS algorithm to match research artifacts across various annotation tools. Although this is effective, LCS has computational limitations, particularly during handling large datasets with complex sentence structures. Investigating more efficient algorithms, dynamic programming techniques, or graph-based mapping approaches, will improve both the speed and accuracy of annotation matching. Additionally, the utilization of transformer-based models for assessing similarity will improve the precision of annotation alignment and reduce errors in cases where artifacts are implied rather than explicitly stated. Furthermore, creating explicit links between Research Artifact annotations and their corresponding Research Artifact Metadata is critically important. The annotation mapping technique that is employed in this study does not include the development of their relationship. This limitation highlights the need for future research to improve the mapping script to support these connections. This will minimize potential information loss and improve integrity within the annotation framework. Additionally, while HITL validation is valuable for ensuring annotation accuracy, it introduces a degree of subjectivity and inconsistency. To address this, future work should explore more automated quality control mechanisms that reduce reliance on manual validation while maintaining a high level of annotation precision.

By expanding beyond the INCEpTION and RAA tools and integrating more advanced computational techniques, future research will enhance the adaptability of annotation mapping workflows. These improvements will contribute to more efficient and



automated NLP annotation processes and make it easier to use across different fields and research groups.



APPENDIX

```
import copy
import pandas as pd
import pylcs
from cassis import *

with open("external_typesystem_master_thesis_vrouvaki_f2822302.xml", "rb") as f:
    typesystem = load_typesystem(f)

with open("daboor_Comparative Analysis of High-Resolution Soil.json", "r",
encoding="utf-8") as f:
    cas = load_cas_from_json(f.read(), typesystem=typesystem)

# Access the sofa string directly from the CAS object

sofa_string = cas.sofa_string
if sofa_string:
    print("Sofa String:", sofa_string)
else:
    print("Sofa String not found in the CAS.")

# Create a copy of the original CAS file for further processing, without affecting the
original one.

CasJson = copy.deepcopy(cas)

file_path = 'raa_tool_example_1.xlsx'
ExcelFile = pd.read_excel(file_path, sheet_name='Mentions')
```



```
# Create new column "Mention_clean" in ExcelFile, which is the "Mention" column,
without <m> and </m>.
```

```
ExcelFile['Mention_clean'] = ExcelFile['Mention'].str.replace('<m>', '', regex=False)
ExcelFile['Mention_clean'] = ExcelFile['Mention_clean'].str.replace('</m>', '',
regex=False)
```

```
ExcelFile['Mention_clean'] = ExcelFile['Mention_clean'].str.replace(r'\s{2,}', ' ',
regex=True)
```

```
# Function to extract sentences from CasJson and store them in a DataFrame.
```

```
def extract_sentences_to_dataframe(CasJson):
```

```
    """
```

```
    Extracts all sentences from the CasJson object and stores them in a DataFrame.
```

```
    Parameters:
```

```
        CasJson (CAS): The JSON-based CAS object.
```

```
    Returns:
```

```
        df_sentences (pd.DataFrame): A DataFrame with columns: 'Sentence', 'begin',
'end'.
```

```
    """
```

```
    Sentence =
```

```
CasJson.typesystem.get_type('de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.
Sentence')
```

```
    sentences = CasJson.select(Sentence)
```

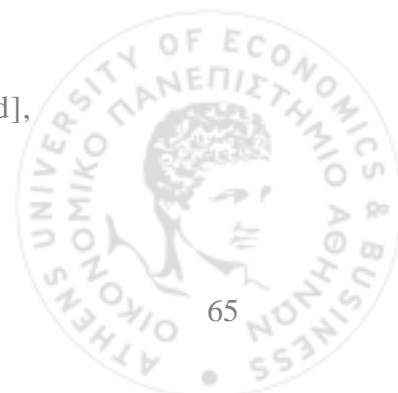
```
    data = [
```

```
        {
```

```
            "Sentence": CasJson.sofa_string[sentence.begin:sentence.end],
```

```
            "begin": sentence.begin,
```

```
            "end": sentence.end
```



```
    }
    for sentence in sentences
]

df_sentences = pd.DataFrame(data)
return df_sentences

df_sentences = extract_sentences_to_dataframe(CasJson)

print("Extracted Sentences:")
print(df_sentences.head())

# Clean the df_sentences: remove "\n\n" from each row and store it in a new column.

df_sentences['Sentence_clean'] = df_sentences['Sentence'].str.replace(r'\n+', ' ',
regex=True)
df_sentences['Sentence_clean'] = df_sentences['Sentence'].str.replace(r'\n', ' ',
regex=True)

df_sentences['Sentence_clean'] = df_sentences['Sentence_clean'].str.replace(r'\s{2,}',
', ', regex=True)
print(df_sentences[['Sentence', 'Sentence_clean']].head())

df_sentences['Sentence_clean'] = df_sentences['Sentence_clean'].str.replace(r'\s+', ' ',
regex=True).str.strip()

### CASE 1:

similarity_results = []
not_found_artifacts_list = []

for index, row in ExcelFile.iterrows():
```



```
research_artifact = row['Research Artifact']
mention_text = row['Mention_clean']
mention_id = row['Mention ID']

filtered_sentences = df_sentences[
    df_sentences["Sentence_clean"].str.contains(research_artifact,
                                                na=False,
                                                regex=False)
]

if filtered_sentences.empty:
    not_found_artifacts_list.append({
        'Research Artifact Index': index,
        'Research Artifact': research_artifact,
        'Mention ID': mention_id,
        'Mention (cleaned)': mention_text
    })
else:
    for _, sentence_row in filtered_sentences.iterrows():
        sentence_text = sentence_row['Sentence_clean']

        lcs_length = pylcs.lcs2(mention_text, sentence_text)

        max_length = max(len(mention_text), len(sentence_text))
        similarity_percentage = (lcs_length / max_length) * 100 if max_length > 0
    else 0

    similarity_results.append({
        'Research Artifact Index': index,
        'Research Artifact': research_artifact,
        'Mention ID': mention_id,
        'Mention (cleaned)': mention_text,
        'Sentence (cleaned)': sentence_text,
        'LCS Length': lcs_length,
```



```
        'Similarity Percentage': similarity_percentage
    })
```

```
df_similarity_results = pd.DataFrame(similarity_results)
not_found_artifacts = pd.DataFrame(not_found_artifacts_list)
```

```
not_found_artifacts_count = not_found_artifacts['Research Artifact Index'].nunique()
not_found_artifacts_count
```

```
if not df_similarity_results.empty:
```

```
    df_similarity_results_count = df_similarity_results['Research Artifact
Index'].nunique()
```

```
    print(f"Number of 'Research Artifact Index' that are matched":
{df_similarity_results_count}")
```

```
else:
```

```
    df_similarity_results_count = 0
```

```
    print("df_similarity_results is empty. Count is set to 0.")
```

```
if not df_similarity_results.empty:
```

```
    max_similarity_results = pd.DataFrame()
```

```
    df_similarity_results_sorted = df_similarity_results.sort_values(by=['Research
Artifact Index', 'Similarity Percentage'], ascending=[True, False])
```

```
    for artifact_index, group in df_similarity_results_sorted.groupby('Research
Artifact Index'):
```

```
        research_artifact = group.iloc[0]['Research Artifact']
```

```
        found = False
```

```
        for _, row in group.iterrows():
```

```
            sentence_cleaned = row['Sentence (cleaned)']
```

```
            if research_artifact in sentence_cleaned:
```



```
max_similarity_results = pd.concat([max_similarity_results,
row.to_frame().T], ignore_index=True)
found = True
break
```

```
if not found:
```

```
not_found_artifacts = pd.concat([not_found_artifacts,
group.iloc[[0]].to_frame().T], ignore_index=True)
```

```
# For those Research Artifacts where a match is found, store the original Json sentence
from the CAS dataset into the original pandas dataframe.
```

```
if not df_similarity_results.empty:
```

```
max_similarity_results['Original Json Sentence'] = None
```

```
for i, row in max_similarity_results.iterrows():
```

```
original_sentence_row = df_sentences[df_sentences['Sentence_clean'] ==
row['Sentence (cleaned)']]
```

```
if not original_sentence_row.empty:
```

```
max_similarity_results.at[i, 'Original Json Sentence'] =
original_sentence_row['Sentence'].values[0]
```

```
if not df_similarity_results.empty:
```

```
merged_df = ExcelFile.merge(max_similarity_results[['Mention ID', 'Original Json
Sentence']], on='Mention ID', how='left')
```

```
ExcelFile['Json Sentence'] = merged_df['Original Json Sentence']
```



CASE 2:

```
ExcelFile['Research Artifact no spaces'] = ExcelFile['Research Artifact'].str.replace(' ', '', regex=False)
```

```
ExcelFile['Mention clean no spaces'] = ExcelFile['Mention_clean'].str.replace(' ', '', regex=False)
```

```
df_sentences['Sentence clean no spaces'] = df_sentences['Sentence_clean'].str.replace(' ', '', regex=False)
```

```
for i, row in ExcelFile[ExcelFile['Json Sentence'].isnull()].iterrows():
```

```
    research_artifact_no_spaces = row['Research Artifact no spaces']
```

```
    mention_clean_no_spaces = row['Mention clean no spaces']
```

```
    matched_sentences = df_sentences[
```

```
        df_sentences['Sentence clean no spaces'].str.contains(research_artifact_no_spaces, na=False)
    ]
```

```
    if not matched_sentences.empty:
```

```
        max_lcs = 0
```

```
        best_sentence = None
```

```
        for _, sentence_row in matched_sentences.iterrows():
```

```
            sentence_no_spaces = sentence_row['Sentence clean no spaces']
```

```
            lcs_length = pylcs.lcs2(mention_clean_no_spaces, sentence_no_spaces)
```

```
            max_length = max(len(mention_clean_no_spaces), len(sentence_no_spaces))
```

```
            similarity_percentage = (lcs_length / max_length) * 100 if max_length > 0
```

```
        else 0
```

```
        if similarity_percentage > max_lcs:
```

```
            max_lcs = similarity_percentage
```

```
            best_sentence = sentence_row['Sentence']
```



```

    if best_sentence:
        ExcelFile.at[i, 'Json Sentence'] = best_sentence

not_found_artifacts = not_found_artifacts[
    not_found_artifacts['Mention ID'].isin(
        ExcelFile[ExcelFile['Json Sentence'].isnull()]['Mention ID']
    )
]

### CASE 3:

filtered_excel = ExcelFile[ExcelFile['Mention ID'].isin(not_found_artifacts['Mention
ID'])]

trigger_similarity_results = []
not_found_triggers_list = []

for index, row in filtered_excel.iterrows():
    trigger_value = row['Trigger']
    mention_text = row['Mention_clean']
    mention_id = row['Mention ID']
    research_artifact = row['Research Artifact']
    research_artifact_index = row.name

    # Filter df_sentences where 'Sentence_clean' contains the 'Trigger' value
    filtered_sentences =
df_sentences[df_sentences['Sentence_clean'].str.contains(trigger_value, na=False,
regex=False)]

if filtered_sentences.empty:
    not_found_triggers_list.append({
        'Research Artifact Index': research_artifact_index,

```



```

        'Research Artifact': research_artifact,
        'Mention ID': mention_id,
        'Mention (cleaned)': mention_text
    })
else:
    for _, sentence_row in filtered_sentences.iterrows():
        sentence_text = sentence_row['Sentence_clean']

        lcs_length = pylcs.lcs2(mention_text, sentence_text)

        max_length = max(len(mention_text), len(sentence_text))
        similarity_percentage = (lcs_length / max_length) * 100 if max_length > 0
    else 0

    trigger_similarity_results.append({
        'Trigger': trigger_value,
        'Mention ID': mention_id,
        'Mention (cleaned)': mention_text,
        'Sentence (cleaned)': sentence_text,
        'LCS Length': lcs_length,
        'Similarity Percentage': similarity_percentage
    })

df_trigger_similarity_results = pd.DataFrame(trigger_similarity_results)
not_found_triggers = pd.DataFrame(not_found_triggers_list)

valid_mention_ids = df_trigger_similarity_results['Mention ID']
df_filtered_trigger_similarity_results = df_trigger_similarity_results[
df_trigger_similarity_results['Mention ID'].isin(valid_mention_ids)]

max_similarity_results_trigger = pd.DataFrame(columns=df_trigger_similarity_results.columns)

```



```

df_trigger_similarity_results_sorted =
df_filtered_trigger_similarity_results.sort_values(
    by=['Mention ID', 'Trigger', 'Mention (cleaned)', 'Similarity Percentage'],
    ascending=[True, True, False, False]
)

for (mention_id, trigger_value, mention_cleaned_value), group in
df_trigger_similarity_results_sorted.groupby(['Mention ID', 'Trigger', 'Mention
(cleaned)']):
    found = False

    for _, row in group.iterrows():
        sentence_cleaned = row['Sentence (cleaned)']

        if sentence_cleaned is not None and trigger_value in sentence_cleaned:
            max_similarity_results_trigger = pd.concat(
                [max_similarity_results_trigger, row.to_frame().T], ignore_index=True
            )
            found = True
            break

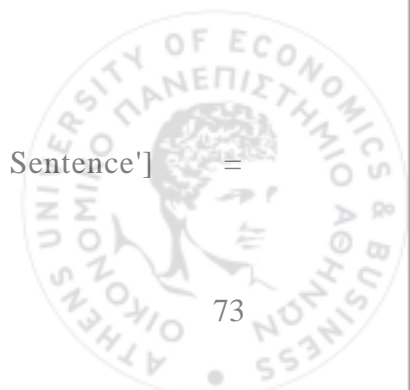
# For those Triggers where a match is found, store the original Json sentence from the
CAS dataset into the original pandas dataframe.

max_similarity_results_trigger['Original Json Sentence'] = None

for i, row in max_similarity_results_trigger.iterrows():
    original_sentence_row = df_sentences[df_sentences['Sentence_clean'] ==
row['Sentence (cleaned)']]

    if not original_sentence_row.empty:
        max_similarity_results_trigger.at[i, 'Original Json Sentence'] =
original_sentence_row['Sentence'].values[0]

```



```

for i, row in ExcelFile.iterrows():
    mention_id = row['Mention ID']

    matching_row =
max_similarity_results_trigger[max_similarity_results_trigger['Mention ID'] ==
mention_id]

    if not matching_row.empty:
        ExcelFile.at[i, 'Json Sentence'] = matching_row['Original Json
Sentence'].values[0]

### CASE 4:

trigger_similarity_results_second_case = []
not_found_triggers_list = []
not_found_triggers_final_list = []

filtered_excel_second_case = ExcelFile[ExcelFile['Mention
ID'].isin(not_found_triggers['Mention ID'])]

for _, row in filtered_excel_second_case.iterrows():
    trigger_value = row['Trigger'].lower()
    mention_text = row['Mention_clean'].lower()
    mention_id = row['Mention ID']

    filtered_sentences = df_sentences[
        df_sentences['Sentence_clean'].str.lower().str.contains(trigger_value, na=False,
regex=False)
    ]

    if filtered_sentences.empty:
        not_found_triggers_list.append({

```



```

        'Trigger': trigger_value,
        'Mention ID': mention_id,
        'Mention (cleaned)': mention_text
    })
else:
    for _, sentence_row in filtered_sentences.iterrows():
        sentence_text = sentence_row['Sentence_clean'].lower()

        lcs_length = pylcs.lcs2(mention_text, sentence_text)

        max_length = max(len(mention_text), len(sentence_text))
        similarity_percentage = (lcs_length / max_length) * 100 if max_length > 0
    else 0

    trigger_similarity_results_second_case.append({
        'Trigger': trigger_value,
        'Mention ID': mention_id,
        'Mention (cleaned)': mention_text,
        'Sentence (cleaned)': sentence_text,
        'LCS Length': lcs_length,
        'Similarity Percentage': similarity_percentage
    })

df_trigger_similarity_results_second_case =
pd.DataFrame(trigger_similarity_results_second_case)
not_found_triggers = pd.DataFrame(not_found_triggers_list)

for _, row in not_found_triggers.iterrows():
    trigger_value = row['Trigger']
    mention_text = row['Mention (cleaned)']
    mention_id = row['Mention ID']

    if not any(df_trigger_similarity_results_second_case['Trigger'] == trigger_value):
        not_found_triggers_final_list.append({

```



```

        'Trigger': trigger_value,
        'Mention ID': mention_id,
        'Mention (cleaned)': mention_text
    })

```

```
not_found_triggers_final = pd.DataFrame(not_found_triggers_final_list)
```

```
max_similarity_results_trigger_second_case = pd.DataFrame()
```

```

df_trigger_similarity_results_sorted =
df_trigger_similarity_results_second_case.sort_values(
    by=['Mention ID', 'Trigger', 'Mention (cleaned)', 'Similarity Percentage'],
    ascending=[True, True, False, False]
)

```

```

for (mention_id, trigger_value, mention_cleaned_value), group in
df_trigger_similarity_results_sorted.groupby(['Mention ID', 'Trigger', 'Mention
(cleaned)']):

```

```
    found = False
```

```
    for _, row in group.iterrows():
```

```
        sentence_cleaned = row['Sentence (cleaned)']
```

```
        if trigger_value in sentence_cleaned:
```

```
            max_similarity_results_trigger_second_case = pd.concat(
```

```
                [max_similarity_results_trigger_second_case, row.to_frame().T],
```

```
            ignore_index=True
```

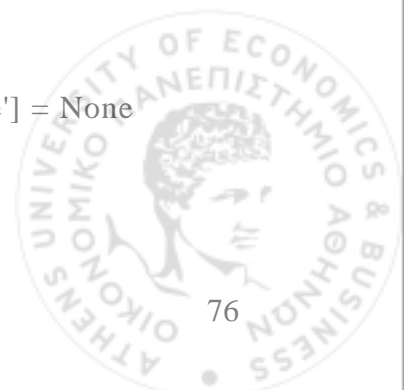
```
            )
```

```
            found = True
```

```
            break
```

```
max_similarity_results_trigger_second_case['Original Json Sentence'] = None
```

```
for i, row in max_similarity_results_trigger_second_case.iterrows():
```



```
sentence_cleaned_lower = row['Sentence (cleaned)'].lower()

original_sentence_row = df_sentences[
    df_sentences['Sentence_clean'].str.lower() == sentence_cleaned_lower
]

if not original_sentence_row.empty:
    max_similarity_results_trigger_second_case.at[i, 'Original Json Sentence'] =
original_sentence_row['Sentence'].values[0]

for i, row in ExcelFile.iterrows():
    mention_id = row['Mention ID']

    matching_row = max_similarity_results_trigger_second_case[
        max_similarity_results_trigger_second_case['Mention ID'] == mention_id
    ]

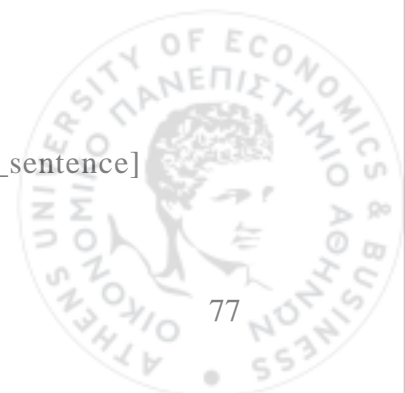
    if not matching_row.empty:
        ExcelFile.at[i, 'Json Sentence'] = matching_row['Original Json
Sentence'].values[0]

# For each Json Sentence that is stored in the original pandas dataframe, take their
corresponding begin and end.

ExcelFile['begin'] = None
ExcelFile['end'] = None

for i, row in ExcelFile.iterrows():
    json_sentence = row['Json Sentence']

    if pd.notnull(json_sentence):
        matched_row = df_sentences[df_sentences['Sentence'] == json_sentence]
```



```

    if not matched_row.empty:
        ExcelFile.at[i, 'begin'] = matched_row['begin'].values[0]
        ExcelFile.at[i, 'end'] = matched_row['end'].values[0]

# Find the initial and final span positions of each Research Artifact or Trigger,
License, Version and URLs.

ExcelFile['span_start'] = None
ExcelFile['span_end'] = None
ExcelFile['license_span_start'] = None
ExcelFile['license_span_end'] = None
ExcelFile['version_span_start'] = None
ExcelFile['version_span_end'] = None
ExcelFile['url_span_start'] = None
ExcelFile['url_span_end'] = None

for index, row in ExcelFile.iterrows():
    json_sentence = row["Json Sentence"]
    research_artifact = row["Research Artifact"]
    trigger = row["Trigger"]
    license = row["License"]
    version = row["Version"]
    url = row["URLs"]
    sentence_begin = row["begin"]

def get_span_positions(term, sentence, sentence_begin):
    """Returns the start and end positions of a term in the sentence."""
    if pd.notna(term) and term in sentence:
        relative_start = sentence.find(term)
        return sentence_begin + relative_start, sentence_begin + relative_start +
len(term)
    elif pd.notna(term):
        term_lower = term.lower()

```



```

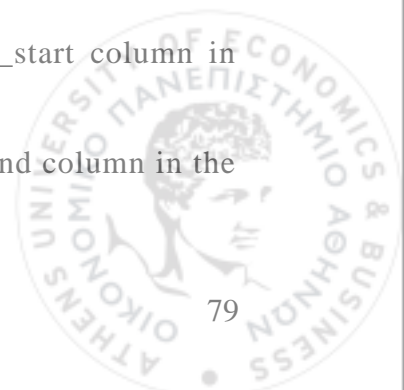
    sentence_lower = sentence.lower()
    if term_lower in sentence_lower:
        relative_start = sentence_lower.find(term_lower)
        return sentence_begin + relative_start, sentence_begin + relative_start +
len(term)
    return None, None
    span_start, span_end = get_span_positions(research_artifact, json_sentence,
sentence_begin)
    trigger_start, trigger_end = get_span_positions(trigger, json_sentence,
sentence_begin)
    license_start, license_end = get_span_positions(license, json_sentence,
sentence_begin)
    version_start, version_end = get_span_positions(version, json_sentence,
sentence_begin)
    url_start, url_end = get_span_positions(url, json_sentence, sentence_begin)

    ExcelFile.at[index, 'span_start'] = span_start if span_start is not None else
trigger_start
    ExcelFile.at[index, 'span_end'] = span_end if span_end is not None else trigger_end
    ExcelFile.at[index, 'license_span_start'] = license_start
    ExcelFile.at[index, 'license_span_end'] = license_end
    ExcelFile.at[index, 'version_span_start'] = version_start
    ExcelFile.at[index, 'version_span_end'] = version_end
    ExcelFile.at[index, 'url_span_start'] = url_start
    ExcelFile.at[index, 'url_span_end'] = url_end

print(ExcelFile["begin"].isna().sum(), "missing values in begin column in the
ExcelFile.")
print(ExcelFile["end"].isna().sum(), "missing values in end column in the ExcelFile.")

print(ExcelFile["span_start"].isna().sum(), "missing values in span_start column in
the ExcelFile.")
print(ExcelFile["span_end"].isna().sum(), "missing values in span_end column in the
ExcelFile.")

```



```
# Extract the TypeSystem from the CAS
typesystem = CasJson.typesystem

print("Available annotation types in TypeSystem:")
for type_name in typesystem.get_types():
    print(type_name)

# Find all instances of FeatureDefinition.

feature_definitions =
list(CasJson.select("de.tudarmstadt.ukp.clarin.webanno.api.type.FeatureDefinition"))

# Filter FeatureDefinitions where the layer name matches
'webanno.custom.ResearchArtifact'.

matching_features = [
    fd for fd in feature_definitions if fd.layer.name ==
"webanno.custom.ResearchArtifact"
]

print("Matching FeatureDefinitions for ResearchArtifact:")
for feature in matching_features:
    print(f"Feature Name: {feature.name}, UI Name: {feature.uiName}")

for sentence in CasJson.select('webanno.custom.ResearchArtifact'):
    for token in
CasJson.select_covered('de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Token
', sentence):
        print(token.get_covered_text())

ExcelFile["span_start"] = ExcelFile["span_start"].astype(int)
ExcelFile["span_end"] = ExcelFile["span_end"].astype(int)
```



```
# Get the ResearchArtifact annotation types.
```

```
ResearchArtifact =  
CasJson.typesystem.get_type("webanno.custom.ResearchArtifact")  
ResearchArtifactMetadata =  
CasJson.typesystem.get_type("webanno.custom.ResearchArtifactMetadata")
```

```
for _, row in ExcelFile.iterrows():
```

```
    if row["Owned"] == "Yes":
```

```
        research_artifact_class = "Owned"
```

```
    elif row["Reused"] == "Yes":
```

```
        research_artifact_class = "Reused"
```

```
    elif "Referenced" in ExcelFile.columns and row["Referenced"] == "Yes":
```

```
        research_artifact_class = "Referenced"
```

```
    else:
```

```
        research_artifact_class = None
```

```
# Create ResearchArtifact annotation.
```

```
new_annotation = ResearchArtifact(  
    ResearchArtifactClass=research_artifact_class,  
    ResearchArtifactType=row["Type"].capitalize() if pd.notna(row["Type"]) else  
None,  
    begin=int(row["span_start"]),  
    end=int(row["span_end"])  
)
```

```
cas.add(new_annotation)
```

```
metadata_references = { }
```

```
metadata_fields = [  
    ("License", "license_span_start", "license_span_end"),
```

```
    ("License", "license_span_start", "license_span_end"),
```



```

("Version", "version_span_start", "version_span_end"),
("URLs", "url_span_start", "url_span_end"),
]

# Create ResearchArtifactMetadata annotation.

for field, start_col, end_col in metadata_fields:
    if pd.notna(row[field]):
        metadata_annotation = ResearchArtifactMetadata(
            begin=int(row[start_col]),
            end=int(row[end_col])
        )
        cas.add(metadata_annotation)
        metadata_references[field] = metadata_annotation

existing_metadata_annotations =
list(cas.select("webanno.custom.ResearchArtifactMetadata"))

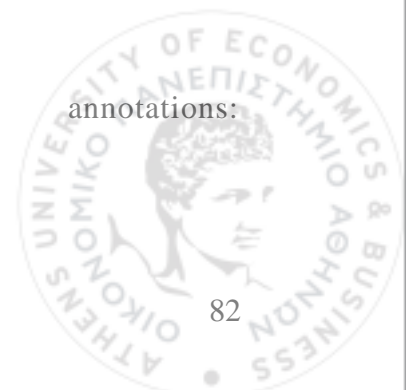
for field, metadata_annotation in metadata_references.items():
    actual_metadata = next(
        (ann for ann in existing_metadata_annotations if ann.begin ==
metadata_annotation.begin and ann.end == metadata_annotation.end),
        None
    )

    if actual_metadata is None:
        print(f" Error: Metadata annotation for {field} not found in CAS!")
        continue

print("Annotations added successfully!")

print(f"Total          ResearchArtifact
{len(list(cas.select('webanno.custom.ResearchArtifact')))}")

```



```
print(f"Total ResearchArtifactMetadata annotations:  
{len(list(cas.select('webanno.custom.ResearchArtifactMetadata')))}")
```

The final CAS file with annotations is then saved in JSON format.

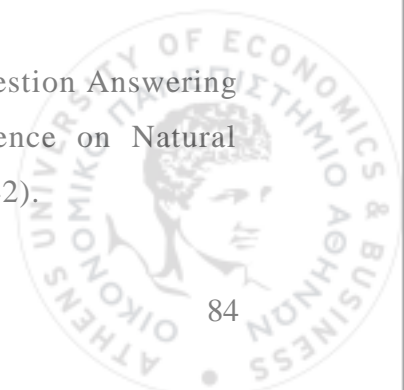
```
cas.to_json('raa_tool_example_1_annotated.json')
```

```
print('Cas File downloaded successfully!')
```



REFERENCES

- [1] Baddour, M., Dameron, O., de Tayrac, M., Paquelet, S., Rollier, P., Labbé, T., & Saleh, M. (2025). ACUIITEE: A Comprehensive Tool for Visualization, Editing and Curating textual Annotations in Clinical Data. In 11th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'25).
- [2] KADAR, M., ADAMACHI, I., & AVRAM, A. PreProcMed: Automated Medical Image Processing Framework for Deep Learning Applications.
- [3] Dunne, C., Shneiderman, B., Gove, R., Klavans, J., & Dorr, B. (2012). Rapid understanding of scientific paper collections: Integrating statistics, text analytics, and visualization. *Journal of the American Society for Information Science and Technology*, 63(12), 2351-2369.
- [4] Hutchinson, T. (2020). Natural language processing and machine learning as practical toolsets for archival processing. *Records Management Journal*, 30(2), 155-174.
- [5] Battogtokh, M., Xing, Y., Davidescu, C., Abdul-Rahman, A., Luck, M., & Borgo, R. (2024, June). Visual Analytics for Fine-grained Text Classification Models and Datasets. In *Computer Graphics Forum* (Vol. 43, No. 3, p. e15098).
- [6] Gambo, I., Massenon, R., Lin, C. C., Ogundokun, R. O., Agarwal, S., & Pak, W. (2024). Enhancing user trust and interpretability in AI-driven feature request detection for mobile app reviews: an explainable approach. *IEEE Access*.
- [7] Chai, C., & Li, G. (2020). Human-in-the-loop Techniques in Machine Learning. *IEEE Data Eng. Bull.*, 43(3), 37-52.
- [8] Peña, A., Morales, A., Fierrez, J., Ortega-Garcia, J., Puente, I., Cordova, J., & Cordova, G. (2024). Continuous document layout analysis: Human-in-the-loop AI-based data curation, database, and evaluation in the domain of public affairs. *Information Fusion*, 108, 102398.
- [9] Kazi, S., & Khoja, S. A. (2024, October). Context-Aware Question Answering in Urdu. In *Proceedings of the 7th International Conference on Natural Language and Speech Processing (ICNLSP 2024)* (pp. 233-242).



- [10] Torri, V., Bottelli, A., Ercolanoni, M., Leoni, O., & Ieva, F. (2025). NLP-based assessment of prescription appropriateness from Italian referrals. arXiv preprint arXiv:2501.14701.
- [11] <https://prodi.gy/>
- [12] Masa, B. A. (2022). Development of an Artificial Intelligence-based Solution for Document Processing Automation Using Machine Learning and NLP Techniques.
- [13] Biemann, C. (2020). ActiveAnno: Flexible and Efficient Document
- [14] <https://github.com/doccano/doccano>
- [15] Naik, V., Patel, P., & Kan86nan, R. (2023). Legal entity extraction: An experimental study of ner approach for legal documents. *International Journal of Advanced Computer Science and Applications*, 14(3).
- [16] <https://webanno.github.io/webanno/>
- [17] Neves, M., & Ševa, J. (2021). An extensive review of tools for manual annotation of documents. *Briefings in bioinformatics*, 22(1), 146-163.
- [18] <https://github.com/iNoBo/scinobo-raa>
- [19] Stavropoulos, P., Lyris, I., Manola, N., Grypari, I., & Papageorgiou, H. (2023, December). Empowering knowledge discovery from scientific literature: a novel approach to research artifact analysis. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)* (pp. 37-53).
- [20] <https://inception-project.github.io/>
- [21] Klie, J. C., Bugert, M., Boullosa, B., De Castilho, R. E., & Gurevych, I. (2018, August). The inception platform: Machine-assisted and knowledge-oriented interactive annotation. In *Proceedings of the 27th international conference on computational linguistics: system demonstrations* (pp. 5-9).
- [22] McNamara, Q., De La Vega, A., & Yarkoni, T. (2017). Developing a comprehensive framework for multimodal feature extraction. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*, 1567–1574. Association for Computing Machinery.
- [23] Jim, J. R., Talukder, M. A. R., Malakar, P., Kabir, M. M., Nur, K., & Mridha, M. F. (2024). Recent advancements and challenges of NLP-based sentiment analysis: A state-of-the-art review. *Natural Language Processing Journal*, 100059.

- [24] Nivetha, K. K., & Priyasadini, S. (2025). Annotate Ease: PDF Metadata Extraction Application Specializing in Research Publications.
- [25] Sewunetie, W. T. (2024). Faculty of Mechanical Engineering and Informatics Extended Sentence Parsing Method for Text-to-Semantic Application (Doctoral dissertation, University of Miskolc).
- [26] Radev, D., Teufel, S., Saggion, H., Lam, W., Blitzer, J., Qi, H., ... & Drabek, E. F. (2003, July). Evaluation challenges in large-scale document summarization. In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (pp. 375-382).
- [27] Abrami, G., & Mehler, A. (2018, May). A UIMA database interface for managing NLP-related text annotations. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018).
- [28] Hong, N., Wen, A., Mojarad, M. R., Sohn, S., Liu, H., & Jiang, G. (2018, December). Standardizing heterogeneous annotation corpora using HL7 FHIR for facilitating their reuse and integration in clinical NLP. In AMIA Annual Symposium Proceedings (Vol. 2018, p. 574).
- [29] Schneider, J. M., Calizzano, R., Kintzel, F., Rehm, G., Galanis, D., & Roberts, I. (2022, June). Towards Practical Semantic Interoperability in NLP Platforms. In Proceedings of the 18th Joint ACL-ISO Workshop on Interoperable Semantic Annotation within LREC2022 (pp. 118-126).
- [30] McCarthy, A. D., Silfverberg, M., Cotterell, R., Hulden, M., & Yarowsky, D. (2018). Marrying universal dependencies and universal morphology. arXiv preprint arXiv:1810.06743.
- [31] <https://www.anaconda.com/docs/tools/anaconda-navigator/main>

