Bratly: A Python Extension for BRAT Functionalities

Jamil Zaghir[*12], **Jean-Philippe Goldman**[*12], **Nikola Bjelogrlic**[*12],
Mina Bjelogrlic[12], **Christian Lovis**[12]
[1] Division of Medical Information Sciences, Geneva University Hospitals, Switzerland
[2] Department of Radiology and Medical Informatics, University of Geneva, Switzerland
jamil.zaghir@unige.ch

Abstract

BRAT is a widely used web-based text annotation tool. However, it lacks robust Python support for effective annotation management and processing. We present Bratly, an open-source extension of BRAT that introduces a solid Python backend, enabling advanced functionalities such as annotation typings, collection typings with statistical insights, corpus and annotation handling, object modifications, and entity-level evaluation based on MUC-5 standards. These enhancements streamline annotation workflows, improve usability, and facilitate high-quality NLP research. This paper outlines the system's architecture, functionalities and evaluation, positioning it as a valuable BRAT extension for its users. The tool is open-source, and the NLP community is welcome to suggest improvements.

1 Introduction

Manual text annotation tools are essential in Natural Language Processing (NLP), as they provide high-quality reference annotations required for training and evaluating models. However, selecting the most suitable tool for a specific annotation project can be challenging due to the large number of available options and the lack of an up-to-date comparison of their features, advantages, and limitations.

A recent study (Neves and Ševa 2021) addressed this challenge by reviewing 78 text annotation tools. To narrow the selection, the authors applied five key criteria: the tool had to be available, web-based, quickly installable (if required), functional for their experiments, and configurable for custom annotation schemes. Based on these criteria, 15 tools, including BRAT (Brat Rapid Annotation Tool) released by Stenetorp et al. (2012), were chosen for an in-depth evaluation.

The evaluation process assessed these tools against 26 criteria spanning publication history, technical specifications, data handling, and functionality. Each criterion was rated on a three-level scale, enabling a systematic comparison and scoring system. The results highlighted differences in tool maturity and comprehensiveness, with scores ranging from 9 to 20. BRAT and WebAnno (Yimam et al. 2013) achieved the highest scores, and emerged as the most effective web-based annotation tools according to the evaluation criteria.

BRAT is designed to enhance the annotation workflow with its intuitive interface and robust visualization of complex annotations. It supports various annotation tasks, including span identification, and binary relations, event annotation, and attribute tagging. As a local, web-based tool built on standard technologies, BRAT's installation process is effortless, and the tool can be configured for diverse annotation needs.

Despite its strengths, BRAT lacks built-in automatic evaluation against a gold-standard dataset (Fort 2016). To address this limitation, we introduce **Bratly**, a Python-based extension that makes the process of analyzing BRAT annotations automatic, efficient and complete for processing large datasets.

2 Bratly functionalities

We do not introduce a stand-alone annotation tool; Bratly is a Python extension layer that operates on BRAT standoff files. It facilitates programmatic work with datasets annotated using BRAT. The core functionalities of Bratly include:

- Annotation typings: A structured schema for managing annotations that leverages

[*] Equal contribution

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 943–949

Pydantic for serialization. BRAT currently supports seven types of annotations: Entity, Relation, Attribute, Normalization, Note, Event, and Equivalence. The Entity type encodes entities within the text, while Relation encodes binary relationships between these entities. The Attribute type allows for the addition of attributes to annotations, such as assigning the attribute "female" to an entity like "wife". Normalization is used to link annotations to concepts in an existing knowledge base. Note enables the creation of comments on annotations or on the document itself. Event describes events triggered by entities, including other involved entities as arguments for the event. Finally, Equivalence establishes equivalence between annotations within the document. Bratly's annotation typings fully support all seven annotation types, including annotated entities that are discontinuous (Figure 1).

- Collection typings and statistics: Tools for managing annotated datasets, computing various statistics, and analyzing the distribution of annotations.
- Input-output functions: Methods for opening, modifying, and saving annotated datasets in the BRAT standoff format.
- Annotation modification utilities: Functions for cleaning and standardizing annotations, including duplicate removal, containment filtering, annotation renumbering, annotation sorting, and selective label removal.
- Entity-level evaluation: A dedicated module that implements the MUC-5 standard for entity-based performance assessment.

These features improve the consistency, quality, and usability of annotated datasets, enabling communities to handle annotations efficiently within Python projects. Table 1 summarizes Bratly's added value compared to BRAT.

3 Repository architecture

The system consists of three primary modules – the last two being installable extras:

- **bratly**: Implements annotation and collection typings, providing the backbone for annotation management. Its Unified Modeling Language (UML) diagrams are depicted in Figure 1 and Figure 2.
- **bratly_io_fs**: Handles corpus reading, writing, allowing seamless interaction with annotation files in BRAT standoff format.
- **bratly_eval**: Implements entity-level evaluation techniques, enabling robust performance analysis against a gold-standard dataset.

The modular design ensures flexibility and ease of use, allowing users to integrate Bratly into their workflows without having to install unnecessary functionalities – with bratly and bratly_io_fs not having any external package dependency. Furthermore, this design allows for the implementation of future modules depending on the needs of the NLP community.

Functionality	**BRAT**	**Bratly**
Span/Entity annotation	Yes (Web)	Yes (API)
Relation/Event annotation	Yes (Web)	Yes (API)
Attribute Tagging	Yes (Web)	Yes (API)
Visualization Interface	Yes	No
Mention search bar	Yes	No
Side-by-side display mode	Yes	No
Syntax checking	Limited (manually, file by file)	Yes
Python annotation typings	No	Yes
Python I/O file-level	No	Yes
Python I/O collection	No	Yes
Annotation type statistics	No	Yes
Annotation label statistics	No	Yes
Entity-level evaluation	No	Yes

Table 1: Comparison of functionalities available on BRAT and Bratly.

4 Implementation Details

Bratly is developed entirely in Python 3.12, with an emphasis on type safety, modularity, and usability. Several implementation choices have been made.

We include Pydantic: all classes benefit from the Pydantic BaseModel functionalities, meaning they inherit Pydantic's data validation and serialization features.

Docker is used to provide a containerized environment for the annotation processing modules. While there is no prebuilt image, the Docker files will be released to the public.

Finally, we use *uv* as the tool for Python package management. This choice is mainly done as it is written in Rust: its performance speed is 31 times faster than *pdm*, and 16 times faster than *poetry*.

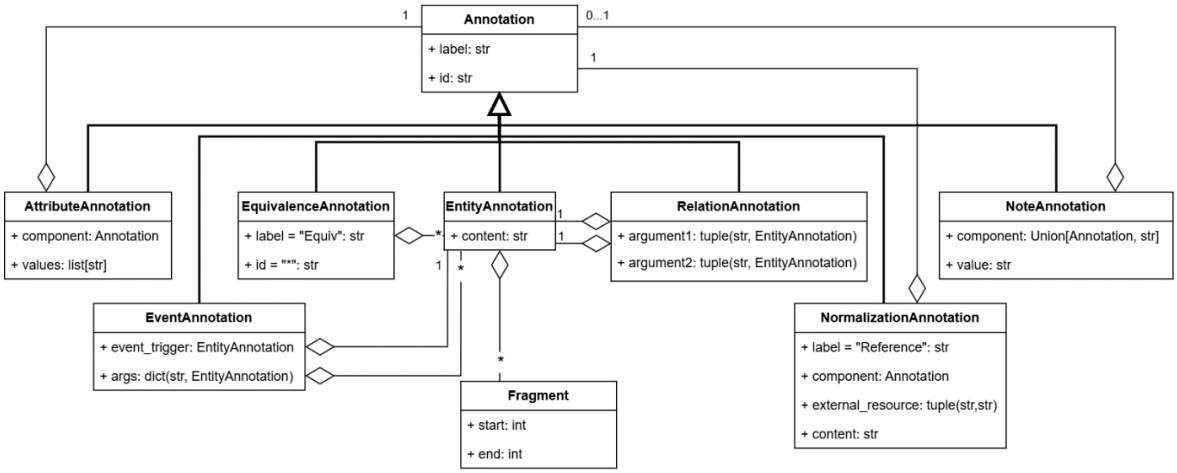

Figure 1: UML Diagram of annotation typings. The abstract class *Annotation* specializes into seven distinct types. *EntityAnnotation* represents entities and can contain one or multiple *Fragment* instances, accommodating discontinuous entities. *RelationAnnotation* encapsulates binary relationships between entities, requiring links to two existing *EntityAnnotation* instances. Both *Annotation* and *Fragment* inherit from Pydantic's BaseModel.

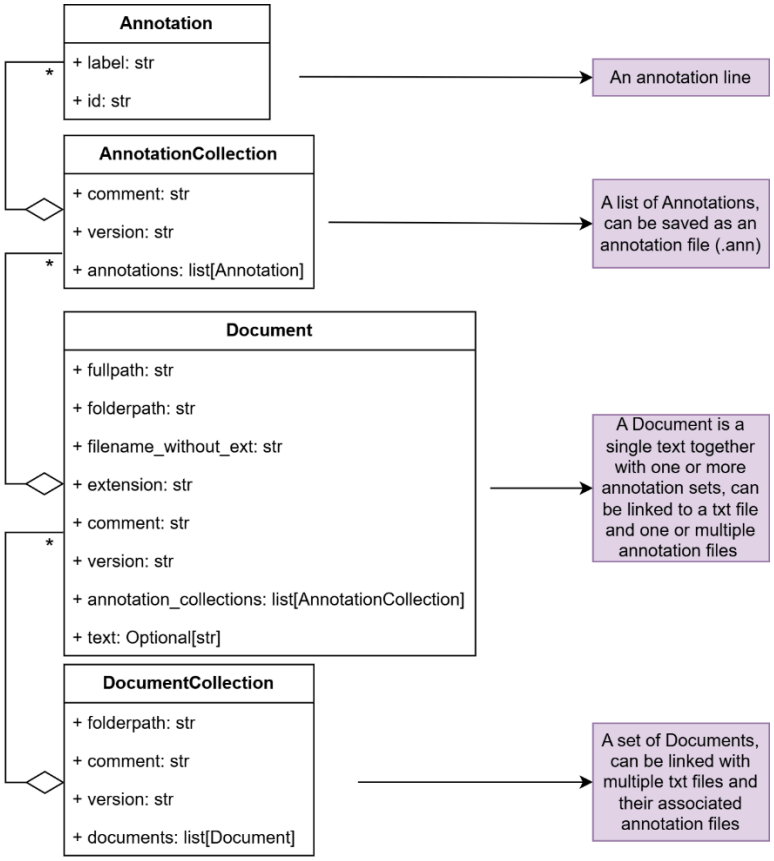

Figure 2: UML Diagram of collection typings. Each Annotation instance represents a single line of annotation. The BRAT annotation file (with an .ann extension) is structured as an AnnotationCollection, containing a list of Annotation instances. A Document includes the path to a textual file (typically a .txt file) and is linked to one or multiple AnnotationCollection instances. This design ensures that a single text file can be annotated by multiple annotation files, facilitating collaborative annotation or encoding different categories of entities separately. DocumentCollection contains a list of Document instances, representing a dataset of annotated files. All four classes inherit from Pydantic's BaseModel.

5 Entity-Level Evaluation

First, Bratly provides entity-level evaluation for Named Entity Recognition (NER). To assess the model's performance fairly, various entity-level metrics exist, such as MUC-5 (Machine Understanding Conference) from Chinchor and Sundheim (1993), ACE (Automatic Content Extraction) from Doddington et al. (2004), and CoNLL (Computational Natural Language Learning) from Tjong Kim Sang and De Meulder (2003). In this project, we choose to implement MUC-5, but the other metrics can be added in the future. The MUC-5 metric comprises:

- CORRECT: Entities accurately identified with matching indices.
- PARTIAL: Partial matches.
- MISSING: Instances where the system fails to identify expected entities.
- SPURIOUS: Predictions not found in the gold standard.

Additionally, Bratly computes entity-level Precision, Recall, and F1-Score, including a Relaxed variant that treats PARTIALs as true predictions alongside CORRECTs (opposed to Strict). A preliminary comparison against token-level NER evaluation shows a notable disparity in model effectiveness when transitioning from token to entity-level metrics, highlighting the importance of entity-level evaluation in NER pipelines (Zaghir et al. 2024).

6 Use case

Processing BRAT-annotated datasets often requires exploring the dataset's statistics, filtering relevant entity annotations, cleaning the data, and preparing it for further analysis.

In this section, we present a use case to filter a dataset to retain only entities labeled as "Organization" and save the refined dataset, enabling one to train a NER model for detecting organizations using this dataset.

As illustrated in Figure 3, the workflow begins by reading a BRAT-annotated dataset into a DocumentCollection, which contains multiple documents with annotated entities and relations.

Then, annotation statistics are computed to analyze the distribution of annotations, helping to understand the dataset's composition. Bratly proposes three levels of statistical analyses: (1) distribution of annotation types, (2) distribution of annotation labels given a particular annotation type, and (3) distribution of textual contents given a particular EntityAnnotation label. In this use case, we used the first two as they are the most relevant for this task.

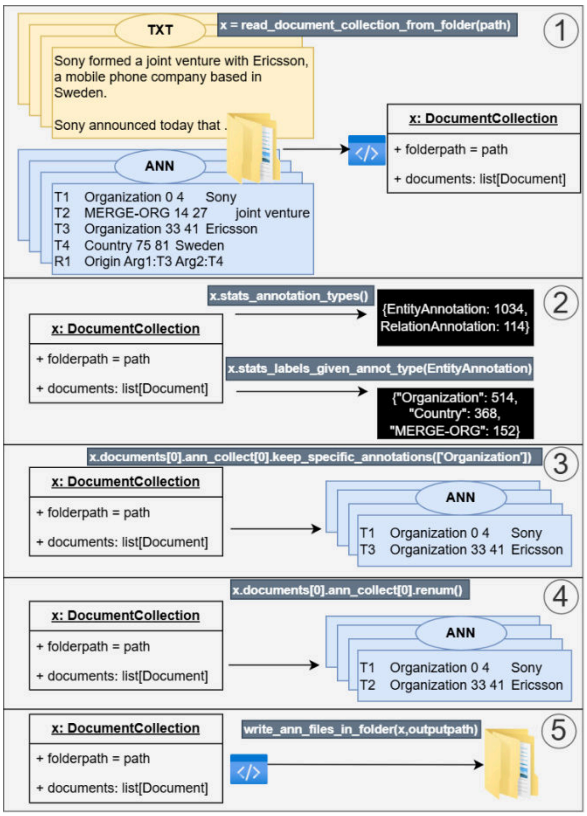

Figure 3: Illustration of a use case using Bratly. 1: Read a BRAT-annotated dataset as a DocumentCollection. 2: Get statistics about annotation types and EntityAnnotation instances in the dataset. 3: Filter in EntityAnnotation instances whose label is Organization. 4: Renumerate annotations. 5: Write the new .ann files in a folder.

A filtering step is then applied to keep only the entity annotations labeled as "Organization", ensuring that other entity types are removed. Once filtered, the annotations are renumbered to maintain a coherent sequence before the processed dataset is exported as .ann files in an output folder whose path is specified as the argument.

The commands in steps 3 and 4 of Figure 3 only modify the first annotation file, demonstrating Bratly's flexibility. This design allows users to selectively edit specific .ann files without affecting others in the dataset. At the same time, applying changes across the entire dataset remains straightforward through looping.

This approach considerably improves dataset management, simplifying the handling of large-scale annotated corpora while ensuring consistency

through various refinement processes. These include annotation renumbering to maintain a coherent sequence, duplicate removal to eliminate redundant entries, and the removal of orphan annotations – such as relations linked to non-existent entities – to preserve the dataset's structural integrity. By facilitating these essential cleaning steps, this method ensures that the data remains well-organized and reliable, making it better suited for downstream applications such as machine learning, linguistic analysis, and other automated annotation processing tasks.

7 Input-output Performance Speed

To evaluate the input-output performance of Bratly, we selected four large datasets annotated with BRAT that are publicly available.

First, CANTEMIST (Miranda-Escalada et al. 2020) is a dataset of Spanish synthetic clinical notes, annotated with tumor morphology entities. Second, Mars (Wagstaff et al. 2018) consists of English abstracts from the Lunar and Planetary Science Conference, covering four Mars missions, with annotations for Minerals, Elements, Properties, and Targets. Third, LSD600 (Nourani et al. 2025) includes English abstracts annotated with diseases and lifestyle factors, and relations between them. Finally, FRASIMED (Zaghir et al. 2024) contains French synthetic clinical cases, annotated with diseases and tumor morphologies.

	CANT EMIST	Mars	LSD	FRASIMED
Entity	16030	94095	13459	24034
Relation	0	10573	2127	0
Equiv.	0	0	274	0
Notes	16030	0	77	36561
Total	32060	104668	15937	60595
NbDocs	1301	1635	600	2051
Read (s)	4.659	17.596	3.426	11.807
Write (s)	0.997	1.539	0.322	1.409

Table 2: Statistics of dataset annotations, including the number of entities, relations, notes, equivalences, and total annotations. NbDocs represents the number of annotated documents. Finally, input-output performance is provided in seconds.

The results are summarized in Table 2. Mars has the highest number of entities (94,095) and relations (10,573), resulting in the largest total annotation count (104,668) and the slowest read time (17.596 seconds). While dataset size and annotation density affect IO performance, writing operations remain consistently fast across all datasets (under 2 seconds). Overall, the processing times remain manageable, demonstrating the efficiency of our package.

8 Discussion

Bratly aims to help advance the field of text annotation tools, particularly for data scientists and researchers in NLP. By extending the widely used BRAT tool through the introduction of a robust Python backend, Bratly addresses several key limitations and enhances the overall functionality, usability, and efficiency of the annotation process. This section discusses the implications of Bratly's features, its potential impact, and future directions for development.

Improved Annotation Management. One of the primary contributions of Bratly is its introduction of structured annotation and collection typings. This structure allows for better organization and management of annotation data. By providing a clear and consistent schema for managing annotations, Bratly helps to maintain high-quality datasets, which is essential for training and evaluating NLP models. The ability to programmatically read, write, and modify annotations further streamlines the annotation workflow, reducing the time and effort required for data preparation. Furthermore, the use of Pydantic for data validation and serialization ensures that Bratly class instances can be easily integrated into larger data pipelines, as Pydantic natively supports JSON handling. The ability to load annotated datasets and compute various statistics provides users with the possibility of exploring their BRAT-annotated datasets, enabling more informed decision-making during the annotation process. As shown in the use case section, Bratly's annotation modification utilities are another key feature that enhances annotation quality. These utilities enable automated cleaning and standardization of annotations, ensuring consistency across the dataset. By automating these processes, Bratly helps maintain a high level of annotation quality and reliability, which should aid in the success of NLP models.

Entity-Level Evaluation Features. Bratly's entity-level evaluation module, based on the MUC-5 criterion, provides a standardized framework for assessing annotation quality. Unlike many existing tools, it includes built-in evaluation features,

enabling the comparison of annotations against a gold-standard dataset using established NLP metrics. This facilitates automatic validation of model-generated annotations, ensuring greater reliability in annotation workflows.

Usability, Accessibility and Expandability. Bratly's input-output functions simplify corpus and annotation handling, making the tool more user-friendly and accessible. The simplified Python API is particularly beneficial for users who may not have extensive programming experience. By lowering the barrier to entry, Bratly enables a broader range of users to leverage its advanced functionalities, thereby promoting wider adoption and utilization of the tool. Furthermore, for advanced researchers interested in improving the package – either privately for their own purposes or publicly by making a pull request through Bratly's Git repository – the modular design of the package offers the required flexibility. Not only does this modularity allow users to integrate Bratly into their existing workflows without disrupting established processes, but it also enables the package to be extended. For example, the annotation class hierarchy (Figure 1) uses the Open-Closed Principle – one of the five SOLID principles in Object-Oriented Programming – allowing for the simple and quick addition of new annotation types in case BRAT releases a new category of annotations.

Future Directions for Bratly. The current state of the package is not meant to be exhaustive, but the required core functionalities are available. There are possible areas for future development and improvement. One possible direction is the expansion of evaluation metrics to include additional standards and criteria. This would provide users with more options for evaluating their annotations. While Bratly has input-output features with the file system through bratly_io_fs, its functionalities could be extended to databases – for example, with an additional module that could be named bratly_io_sql.

9 Python package installation and GitHub repository

The GitHub repository is available at the following link: https://github.com/SimedDataTeam/bratly/.

As mentioned in Section 3, Bratly is modular and can be installed with extras through PyPi:

- **pip install bratly**: bratly package alone.
- **pip install bratly[io]**: bratly and bratly_io_fs packages.
- **pip install bratly[eval]**: bratly, bratly_io_fs, and bratly_eval packages.

10 Conclusion

In conclusion, Bratly is a Python-backed extension of the Brat tool, addressing several key limitations and enhancing the overall functionality, usability, and efficiency of the annotation process. By introducing structured annotation typings and advanced evaluation features, Bratly aims to enable BRAT users to conduct more efficient and scalable annotation tasks, thereby improving the quality and reliability of annotated datasets. We hope the potential impact of Bratly will be significant, as it provides the required tools to manage and evaluate BRAT annotations effectively. Its open-source nature welcomes any future contributions from the NLP community.

References

Chinchor, Nancy, and Beth Sundheim. 1993. "MUC-5 Evaluation Metrics." In *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27*. https://doi.org/10.3115/1072017.1072026.

Doddington, George R., Alexis Mitchell, Mark A. Przybocki, Lance A. Ramshaw, Stephanie M. Strassel, and Ralph M. Weischedel. 2004. "The Automatic Content Extraction (ACE) Program Tasks, Data, and Evaluation." In *LREC*, 2:837–40. Lisbon. http://lrec.elra.info/proceedings/lrec2004/pdf/5.pdf.

Fort, Karën. 2016. *Collaborative Annotation for Reliable Natural Language Processing: Technical and Sociological Aspects*. John Wiley & Sons.

Miranda-Escalada, Antonio, Eulàlia Farré, and Martin Krallinger. 2020. "Named Entity Recognition, Concept Normalization and Clinical Coding: Overview of the Cantemist Track for Cancer Text Mining in Spanish, Corpus, Guidelines, Methods and Results." *IberLEF@ SEPLN*, 303–23.

Neves, Mariana, and Jurica Ševa. 2021. "An Extensive Review of Tools for Manual Annotation of Documents." *Briefings in Bioinformatics* 22 (1): 146–63. https://doi.org/10/ggqtkq.

Nourani, Esmaeil, Evangelia-Mantelena Makri, Xiqing Mao, Sampo Pyysalo, Søren Brunak, Katerina Nastou, and Lars Juhl Jensen. 2025. "LSD600: The First Corpus of Biomedical Abstracts Annotated with Lifestyle–Disease Relations." *Database* 2025 (January):baae129. https://doi.org/10.1093/database/baae129.

Stenetorp, Pontus, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. "BRAT: A Web-Based Tool for NLP-Assisted Text Annotation." In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 102–7.

Tjong Kim Sang, Erik F., and Fien De Meulder. 2003. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition." In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 142–47. https://aclanthology.org/W03-0419.

Wagstaff, Kiri, Raymond Francis, Thamme Gowda, You Lu, Ellen Riloff, Karanjeet Singh, and Nina Lanza. 2018. "Mars Target Encyclopedia: Rock and Soil Composition Extracted From the Literature." *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (1). https://doi.org/10.1609/aaai.v32i1.11412.

Yimam, Seid Muhie, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. 2013. "WebAnno: A Flexible, Web-Based and Visually Supported System for Distributed Annotations." In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, edited by Miriam Butt and Sarmad Hussain, 1–6. Sofia, Bulgaria: Association for Computational Linguistics. https://aclanthology.org/P13-4001/.

Zaghir, Jamil, Mina Bjelogrlic, Jean-Philippe Goldman, Soukaïna Aananou, Christophe Gaudet-Blavignac, and Christian Lovis. 2024. "FRASIMED: A Clinical French Annotated Resource Produced through Crosslingual BERT-Based Annotation Projection." In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, edited by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, 7450–60. Torino, Italia: ELRA and ICCL. https://aclanthology.org/2024.lrec-main.657/.

Zaghir, Jamil, Mina Bjelogrlic, Jean-Philippe Goldman, Adel Bensahla, Yuanyuan Zheng, and Christian Lovis. 2024. "Beyond Tokens: Fair Evaluation of French Large Language Models for Clinical Named Entity Recognition." *Studies in Health Technology and Informatics* 316 (August):666–70. https://doi.org/10.3233/SHTI240502.