



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT

FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN



DISSERTATION

Natural Language Processing in Large-Scale Text Collections for the Digital Humanities & Computational Social Science

Fynn Petersen-Frey

Language Technology

Department of Informatics

Faculty of Mathematics, Informatics and Natural Sciences

Universität Hamburg

Hamburg, Germany

A dissertation submitted for the degree of

Doctor rerum naturalium (Dr. rer. nat.)

2026

Natural Language Processing in Large-Scale Text Collections for the Digital Humanities & Computational Social Science

Dissertation submitted by: Fynn Petersen-Frey

Date of Submission: 21.04.2026

Date of Disputation: 23.06.2026

Supervisor: Prof. Dr. Chris Biemann, Universität Hamburg

Committee:

1st Reviewer: Prof. Dr. Chris Biemann, Universität Hamburg

2nd Reviewer: Prof. Dr. Torsten Zesch, FernUniversität in Hagen

Chair: Prof. Dr. Janick Edinger, Universität Hamburg

Co-Chair: Prof. Dr. Ralf Möller, Universität Hamburg

Universität Hamburg

Faculty of Mathematics, Informatics and Natural Sciences

Department of Informatics

Language Technology

Affidavit

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Sofern im Zuge der Erstellung der vorliegenden Dissertationsschrift generative Künstliche Intelligenz (gKI) basierte elektronische Hilfsmittel verwendet wurden, versichere ich, dass meine eigene Leistung im Vordergrund stand und dass eine vollständige Dokumentation aller verwendeten Hilfsmittel gemäß der Guten wissenschaftlichen Praxis vorliegt. Ich trage die Verantwortung für eventuell durch die gKI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

Affidavit

I hereby declare and affirm that this doctoral dissertation is my own work and that I have not used any aids and sources other than those indicated. If electronic resources based on generative artificial intelligence (gAI) were used in the course of writing this dissertation, I confirm that my own work was the main and value-adding contribution and that complete documentation of all resources used is available in accordance with good scientific practice. I am responsible for any erroneous or distorted content, incorrect references, violations of data protection and copyright law or plagiarism that may have been generated by the gAI.

Ort, Datum | City, Date

Unterschrift | Signature

(Fynn Petersen-Frey)

Declaration of Third-Party Tools

Several software tools have been used to create this dissertation:

I wrote this dissertation in $\text{\LaTeX}$ using the VSCode editor¹ with the Visual Studio Code LaTeX Workshop Extension² that integrates LanguageTool³ to check for grammatical errors and spelling mistakes.

To create figures, I mainly used draw.io⁴. Some figures are generated from data with Matplotlib⁵, and a few figures were created using TikZ⁶. The Figures 6.1, 6.4 and 6.5 showing colorful coreference annotations were created with my own visualization and analysis tool.

No generative artificial intelligence, large language model (LLM) resp. generative pre-trained transformer (GPT) has been used to write this dissertation except generating the example texts for Figures 4.1 and 4.2 with the LLM gemma-3-27b-it⁷ (Team et al., 2025) using a self-hosted Open WebUI⁸ instance (Baek et al., 2025).

¹<https://vscodium.com> and <https://github.com/VSCodium/vscodium>

²<https://github.com/James-Yu/LaTeX-Workshop>

³<https://languagetool.org>

⁴<https://draw.io> resp. <https://app.diagrams.net>

⁵<https://matplotlib.org>

⁶<https://tikz.net>

⁷<https://huggingface.co/google/gemma-3-27b-it>

⁸<https://openwebui.com> and <https://github.com/open-webui/open-webui>

To my family, with love and gratitude — filling my heart with joy every day.

Acknowledgements

I would like to say *thank you* to all people who believed in me, supported me and encouraged me to start, advance and finish this dissertation project. This work was only possible due to the constant support, motivation and guidance of several important individuals — *thank you!*

First and foremost, I would like to thank my supervisor, Chris. Your guidance, patience and expertise have been absolutely invaluable to me and this dissertation project. I am most grateful for your incredibly constructive and actionable feedback on many things, for your understanding of my complexly-structured parental leave, and unwavering support especially during challenging times of which there were more than enough with a young child. Finally, I thank you for persistently encouraging and driving me to complete this dissertation!

I would also like to thank Martin for regularly encouraging me to finish the dissertation, and giving me the opportunity to focus mostly on my dissertation to get it over the finish line.

I would like to express my gratitude to my (former) colleagues and collaborators for your contributions to this dissertation in form of inspiring discussions, collaborations on projects and publications, as well as brainstorming and planning sessions on a wealth of challenging and fascinating topics such as the nitty-gritty details of coreference resolution, neural networks, and DATS. Thank you also for making our office such a nice place to be with lively breaks, random chats and awesome events, as well as making memorable conference trips together, and having a blast at the final Informatik:UM party!

Additionally, I want to thank my friends for their general interest, motivation, curious questions, and the many extremely welcome distractions. Thanks also for regularly asking when my dissertation will finally be finished.

I would like to express my heartfelt gratitude to my parents. Your unwavering belief, support, love and encouragement throughout my entire life has enabled me to pursue this academic journey and reach this point of finishing my dissertation. Danke Renate, danke Olaf!

Finally, I would like to express my deepest gratitude to my wife, Swaantje. Your presence in my life is a constant source of joy, motivation and strength for me! Your understanding, unwavering belief in me, inexhaustible patience and infinite love have been indispensable to me, and helped me through this and other demanding journeys. I am deeply grateful that you took on so many things to provide me with the time and focus for finishing this dissertation. I am forever grateful for your invaluable support.

Abstract

Humans and machines produce ever-growing amounts of data, including text written in natural languages, at an increasing rate every day. In the Digital Humanities (DH) and Computational Social Science (CSS), the sheer amount of textual data is impossible to work through manually. For example, when analyzing the effects of climate change on societies, researchers have to analyze a variety of text types in large quantities such as news articles, (non-)governmental reports, law texts, regulations, contracts, and social media data. Thus, automation with natural language processing (NLP) is essential to deal with large-scale text collections.

The DH are an interdisciplinary area of research combining humanities and computer science by applying computational methods to humanities research, often using computing methods to produce large-scale digital text collections or analyze them to gain an understanding of the researched phenomenon. CSS is an interdisciplinary field at the intersection of social science and computer science that tests and develops scientific theories about human behavior by applying computational methods on large, often textual data from the internet.

NLP is a subfield of computer science that deals with the processing of natural language by computers, and is nowadays closely connected with machine learning (ML) and artificial intelligence (AI). NLP approaches have been applied to many DH or CSS use cases by supporting the exploration and analysis of textual data at scale, but require integration into easily applicable software to make them accessible to domain experts. Although NLP has seen groundbreaking success with pre-trained language models (PLMs) and large language models (LLMs), this development is often not reflected in DH or CSS to the same degree due to black-box behavior, limited applicability, or lacking availability in ready-to-use software.

To improve this situation, this dissertation has the goal to bring state of the art NLP methods for large-scale text collections to DH and CSS. First, we review available NLP-assisted research tools and methods for DH and CSS. Second, we develop novel NLP methods based on neural PLMs that are broadly applicable in many DH and CSS contexts; to assist researchers with large-scale text collections to answer their research questions. Third, we develop a suitable research platform architecture, implement necessary key functionalities, and integrate the NLP methods into the user-friendly research platform.

We describe methods using PLMs and their contextualized word embeddings to improve semantic similarity search for DH, detect and attribute quotations, perform coreference resolution in German, and semi-automatically scale manual annotations. First, we present a configurable similarity search providing insights into which aspects of documents are similar, and a method to search for sub-sentence text spans using semantic search as a modern alternative to KWIC (Key Word in Context). Second, we work on quotation detection and attribution by creating a new manually annotated dataset, and develop a novel ML model performing high-quality quotation detection and attribution with vast improvements over other systems. Third, we present two neural coreference resolution models for German that improve substantially on previous systems while being computationally efficient even for long documents. Last, we show a way to semi-automatically scale manual annotations to large corpora by designing a suitable user interface and developing an interactive annotation method based on contextualized word embeddings and k -nearest neighbors.

To develop the platform, we perform a requirements analysis to derive the design goals for the platform, and develop suitable architectures for all parts of the platform to enable a maintainable easy-to-use software for working on a wide variety of research questions with large-scale text collections. We present crucial implementation details, and explain the integration of the NLP methods into the platform.

We find that our developed methods are widely applicable to many research questions as they are agnostic to the individual use case, can be used both in isolation and combination with other methods. Thereby, researchers can build upon the results of our methods to perform their specific analysis. We find that a suitable software architecture is needed to integrate various NLP methods, both interactive and long-running tasks, with a unified data model to improve the compositionality of the methods. Our developed platform is a suitable foundation to make various NLP methods available to end users. It already integrates a wide range of novel NLP methods with support for large data quantities and provides an easy-to-use interface for DH and CSS researchers, thereby enabling them to analyze their data to answer their individual research questions.

Overall, this dissertation brings state of the art NLP methods for analyzing large-scale text collections with variety of research questions to DH and CSS by providing broadly-applicable methods in an easy-to-use software platform capable of handling large-scale data.

Zusammenfassung

Täglich werden immer größere Datenmengen, wie beispielsweise Texte in natürlicher Sprache, mit steigender Geschwindigkeit produziert. Auch in den digitalen Geisteswissenschaften (*Digital Humanities*, DH) und computergestützten Sozialwissenschaften (*Computational Social Science*, CSS) ist die schiere Menge an Textdaten manuell unmöglich zu verarbeiten. Beispielsweise müssen Forschende bei der Analyse der Auswirkungen des Klimawandels auf Gesellschaften eine Vielzahl von Textarten in großen Mengen analysieren, u.a. Nachrichtenartikel, staatliche Berichte, Gesetzestexte, Vorschriften, Verträge und Daten aus sozialen Medien.

Daher ist eine Automatisierung mittels der automatischen Verarbeitung natürlicher Sprache (*natural language processing*, NLP) essentiell um mit großen Textsammlungen zu arbeiten.

Die DH sind ein interdisziplinäres Forschungsgebiet, das Geisteswissenschaften und Informatik verbindet, indem es computergestützte Methoden in geisteswissenschaftlicher Forschung anwendet. Dabei werden oft Methoden eingesetzt um große digitale Textsammlungen zu erstellen oder diese zu analysieren, um das untersuchte Phänomen zu verstehen. CSS ist ein interdisziplinäres Forschungsfeld zwischen Sozialwissenschaft und Informatik, welches wissenschaftliche Theorien über das menschliche Verhalten entwickelt und testet, indem computergestützte Methoden auf große zumeist Textdaten aus dem Internet angewandt werden.

NLP ist eine Teildisziplin der Informatik, die untersucht wie natürliche Sprache mit Computern verarbeitet werden kann, wobei heutzutage oft maschinelles Lernen (ML) und künstliche Intelligenz zum Einsatz kommen. NLP-Ansätze wurden in vielen DH oder CSS Anwendungsfällen genutzt, um Textdaten zu untersuchen und die Textdatenanalyse zu skalieren. Jedoch müssen diese in nutzbare Software integriert werden, damit sie für die Fachexperten zugänglich sind. Obwohl NLP mit vortrainierten Sprachmodellen (*pre-trained language models*, PLMs) bzw. großen Sprachmodellen (*large language models*, LLMs) bahnbrechende Erfolge erzielt hat, spiegelt sich diese Entwicklung oftmals aufgrund des Black-Box-Verhaltens, der begrenzten Anwendbarkeit oder der mangelnden Verfügbarkeit in nutzbarer Software nicht in gleichem Maße in DH oder CSS wider.

Um diese Situation zu verbessern, hat diese Dissertation das Ziel, modernste NLP-Methoden für große Textsammlungen in die DH und CSS zu bringen. Erstens wird ein Überblick über verfügbare NLP-gestützte Forschungswerkzeuge und -methoden für die DH und CSS gegeben. Zweitens werden neuartige NLP-Methoden auf Basis neuronaler PLMs entwickelt, die in vielen DH- und CSS-Kontexten einsetzbar sind, um Forschern mit umfangreichen Textsammlungen bei der Beantwortung ihrer Forschungsfragen zu helfen. Drittens wird eine geeignete Forschungsplattformarchitektur entwickelt, die erforderlichen zentralen Funktionen implementiert und die NLP-Methoden in die benutzerfreundliche Software integriert.

In dieser Arbeit werden Methoden beschrieben, die auf PLMs und deren kontextualisierten Wort-Embeddings (*contextualized word embeddings*, CWEs) basieren, um die semantische Ähnlichkeitssuche für die DH zu verbessern, Zitate zu erkennen und zuzuordnen, die Koreferenzauflösung auf Deutsch durchzuführen und manuelle Annotationen halbautomatisch zu skalieren. Erstens zeigt diese Arbeit eine konfigurierbare Ähnlichkeitssuche, die Aufschluss darüber gibt, welche Aspekte von Dokumenten ähnlich sind, und eine Methode zur Suche nach Textabschnitten innerhalb von Sätzen unter Verwendung der semantischen Suche als

moderne Alternative zu KWIC (*Key Word in Context*, Schlüsselwörter im Kontext). Zweitens wird die Erkennung und Zuordnung von Zitaten automatisiert, indem ein neuer manuell annotierter Datensatz erstellt und ein neuartiges ML-Modell entwickelt wird, das eine hochwertige Erkennung und Zuordnung von Zitaten ermöglicht, wobei es gegenüber anderen Systemen erhebliche Verbesserungen aufweist. Drittens werden zwei neuronale Modelle zur Auflösung von Koreferenzen für Deutsch vorgestellt, die gegenüber vorherigen Systemen enorme Verbesserungen aufweisen und selbst bei langen Dokumenten effizient zu berechnen sind. Viertens zeigt diese Arbeit eine Möglichkeit manuelle Annotationen mit einer grafischen Anwendung und interaktiven Methode auf der Grundlage von contextualized word embeddings und k -NN (*k-nearest neighbors*, k -nächste-Nachbarn) halbautomatisch auf große Korpora zu skalieren.

Zur Entwicklung der Plattform führen wir eine Anforderungsanalyse durch, um daraus die Designziele für die Plattform abzuleiten, und entwickeln geeignete Architekturen für alle Teile der Plattform, um eine wartbare, benutzerfreundliche Software für die Bearbeitung einer Vielzahl von Forschungsfragen mit umfangreichen Textsammlungen zu ermöglichen. Wir stellen wichtige Details zur Implementierung vor und erläutern die Integration der NLP-Methoden in die Plattform.

Wir haben festgestellt, dass unsere entwickelten Methoden auf viele Forschungsfragen anwendbar sind, da sie unabhängig vom jeweiligen Anwendungsfall sind und sowohl isoliert als auch in Kombination mit anderen Methoden eingesetzt werden können. Dadurch können Forschende auf den Ergebnissen der Methoden aufbauen, um ihre spezifischen Analysen durchzuführen. Wir haben gesehen, dass eine geeignete Softwarearchitektur erforderlich ist, um verschiedene NLP-Methoden, sowohl interaktive als auch langlaufende Vorgänge, mit einem einheitlichen Datenmodell zu integrieren, um die Kompositionalität der Methoden zu verbessern. Unsere entwickelte Plattform bietet eine geeignete Grundlage für die Integration einer Vielzahl von NLP-Methoden mit Unterstützung für große Datenmengen und einer benutzerfreundlichen Oberfläche für DH- und CSS-Forschende, wodurch diese ihre Daten analysieren können, um individuelle Forschungsfragen zu beantworten.

Insgesamt bringt diese Dissertation modernste NLP-Methoden zur Analyse umfangreicher Textsammlungen mit einer Vielzahl von Forschungsfragen in die DH und CSS, indem sie allgemein anwendbare Methoden in einer benutzerfreundlichen Softwareplattform bereitstellt, die große Datenmengen verarbeiten kann.

Contents

List of Abbreviations	VII
List of Figures	VIII
List of Tables	XI

I	Introduction, Background & Related Work	1
1	Introduction	3
1.1	NLP for Digital Humanities and Computational Social Science	4
1.2	Research Questions	5
1.3	At the Intersection of Social Science, NLP and Software Engineering	5
1.4	Publications	6
1.4.1	Dissertation Related Publications	6
1.4.2	Other Publications	9
1.5	Overview & Content Structure	9
2	Background	11
2.1	Machine Learning	11
2.1.1	Fundamentals	12
2.1.2	Evaluation	14
2.2	Neural Networks	16
2.2.1	Basics	16
2.2.2	Feedforward Neural Networks	19
2.2.3	Inference	19
2.2.4	Training	20
2.2.5	Recurrent Neural Networks	23
2.2.6	Gated Recurrent Neural Networks	25
2.2.7	Transformers	26
2.3	Natural Language Processing	28
2.3.1	Text as Symbols	28
2.3.2	Word Embeddings	29
2.3.3	Pre-trained Language Models	29
3	General Related Work	31
3.1	NLP-assisted Research Software in DH and CSS	31
3.1.1	NLP Annotation Tools for Linguistic Annotations	31
3.1.2	DH and CSS Annotation and Analysis Tools	33
3.2	NLP Methods in the Digital Humanities and Computational Social Science	37
3.2.1	Data Exploration	38
3.2.2	Search	39
3.2.3	Extracting Structured Information	40

II	Broadly Applicable NLP for DH & CSS Workflows	43
4	Semantic Similarity Search for the Digital Humanities	45
4.1	Introduction	46
4.1.1	What is Similarity in Embeddings	46
4.1.2	Providing Explainability	47
4.1.3	Goal and Outline	47
4.2	Related Work	47
4.2.1	General Document or Sentence Similarity	47
4.2.2	Aspect-based Similarity	48
4.2.3	Structured Similarity	49
4.3	Configurable Similarity Search	49
4.3.1	Experiments using the SemEval 2022 Task 8 Dataset	51
4.3.2	Application-emulating Experiments	55
4.4	Structured Similarity Search	58
4.4.1	Methodology	58
4.4.2	Linguistic Structures	60
4.4.3	Experiments	61
4.4.4	Results	62
4.5	Conclusion	64
5	Quotation Detection and Attribution	67
5.1	Introduction	68
5.1.1	Task Description	68
5.1.2	Use Cases	69
5.1.3	Chapter Overview	70
5.2	Related Work	70
5.2.1	Quotation Detection and Attribution	70
5.2.2	Semantic Role Labeling	72
5.2.3	Structured Generation	72
5.3	News Article Dataset with Quotation Annotations	72
5.3.1	Annotation Schema	74
5.3.2	Dataset Creation	77
5.3.3	Quantitative Analysis	81
5.4	Systems	84
5.4.1	QUiTE: A rule-based System	84
5.4.2	Citron	86
5.4.3	Quotect: Fine-grained Quotation Detection and Attribution	86
5.5	Experiments	93
5.5.1	Evaluation Metrics	93
5.5.2	Data	94
5.5.3	Quotect Model Variants	94
5.5.4	Training Details	94
5.5.5	Main Results	95
5.5.6	QUiTE and Citron Ablation Study	98

5.5.7	Quotect Model Variants Ablation Study	99
5.5.8	Inference Runtime Efficiency	101
5.5.9	Error Analysis	102
5.6	Conclusion	102
5.6.1	Ethical Considerations	103
5.6.2	Limitations	104
5.6.3	The Case for Coreference Resolution	104
6	Transformer-based Coreference Resolution for German	105
6.1	Introduction	106
6.2	Related Work	107
6.2.1	Coreference Resolution for English	107
6.2.2	Coreference Resolution for German	109
6.3	Datasets	109
6.4	Models	110
6.4.1	Coarse-to-Fine	111
6.4.2	Maverick	113
6.4.3	Model Architecture Comparison	115
6.5	Experiments	116
6.5.1	Evaluation	116
6.5.2	Preliminary Model Design Experiments: Coarse-to-Fine	117
6.5.3	Preliminary Model Design Experiments: Maverick	118
6.5.4	Main Experiments	119
6.5.5	Error Analysis	121
6.6	Conclusion	122
6.6.1	Ethical Considerations	123
6.6.2	Limitations	124
7	Scaling Manual Annotations to a Large Corpus	125
7.1	Introduction	125
7.2	Related Work	127
7.2.1	Natural Language Processing Methods for few Training Examples	127
7.2.2	Existing Annotation Software	128
7.3	Application	128
7.3.1	Requirements	128
7.3.2	Workflow and User Interface	129
7.4	Method	131
7.4.1	Pre-processing	131
7.4.2	Best-matching Structure Selection	131
7.4.3	Interactive, Semi-automatic Annotation Scaling	133
7.4.4	Fully-automatic Annotation Scaling for Quantitative Analysis	134
7.5	Experiments	134
7.5.1	Datasets	135
7.5.2	Semi-automatic Annotation Scaling	135
7.6	Conclusion	137

III	Platform, NLP Method Integration & Conclusion	139
8	Platform	141
8.1	Introduction	142
8.2	Requirements Analysis	142
8.2.1	CLICCS B1 Media Project	142
8.2.2	D-WISE Project	143
8.2.3	HCDS	143
8.2.4	Issues with Existing Software and Methods	144
8.2.5	Common Issues with Research Software	144
8.3	Design Goals	145
8.3.1	Modularity	145
8.3.2	Usability	145
8.3.3	Configurability	146
8.3.4	Widely Applicable	146
8.3.5	Interoperability	146
8.3.6	Large-scale Data Performance	147
8.3.7	Large-scale Data Methodology	147
8.3.8	Interactivity	148
8.3.9	Collaboration	149
8.3.10	Deployment	149
8.4	Architecture	150
8.4.1	Overview	150
8.4.2	Client-Server Communication	151
8.4.3	Server	152
8.4.4	Client	155
8.4.5	Summary	157
9	Applying the NLP methods in the Platform	159
9.1	Introduction	160
9.1.1	DWTS and DATS	160
9.1.2	Contributions	161
9.1.3	Chapter Overview	161
9.2	Implementation Aspects	162
9.2.1	Tech Stack	162
9.2.2	Data Model	162
9.2.3	Services	163
9.2.4	Deployment	163
9.2.5	Annotation Viewer and Efficient Manual Annotation	164
9.2.6	Database Migrations	166
9.2.7	Job System	167
9.3	Semantic Similarity Search	169
9.3.1	Frontend	169
9.3.2	Backend	171

9.4	Coreference Resolution	172
9.4.1	Frontend: Visualizing Coreference Annotations in Documents	172
9.4.2	Backend	172
9.5	Fine-grained Quotation Detection and Attribution	174
9.5.1	Frontend	174
9.5.2	Backend	174
9.5.3	Combining Quotation and Coreference Information	174
9.6	Annotation Scaling	175
9.6.1	Frontend	175
9.6.2	Backend	176
10	Conclusion	179
10.1	New NLP Methods for DH & CSS Workflows	179
10.1.1	Semantic Similarity Search for the Digital Humanities	179
10.1.2	Quotation Detection and Attribution	180
10.1.3	Transformer-based Coreference Resolution for German	180
10.1.4	Scaling Manual Annotations to a Large Corpus	181
10.2	Platform and NLP Method Application	181
10.2.1	Platform	182
10.2.2	Applying the NLP Methods in the Platform	182
10.3	Answering the Research Questions	183
10.4	Outlook	184
10.4.1	Envisioned Extensions of the Platform	184
10.4.2	Future Directions for NLP Methods in DH and CSS	185
	Bibliography	187

List of Abbreviations

AGI	artificial general intelligence
AI	artificial intelligence
AL	active learning
ANN	approximate nearest neighbors
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BERT	Bidirectional Encoder Representations from Transformers
BPTT	backpropagation through time
CAQDAS	computer assisted qualitative data analysis software
CLICCS	Climate, Climate Change, and Society
CNN	convolutional neural network
CPU	Central Processing Unit
CSS	Computational Social Science
CSV	Comma-Separated Values
CWE	contextualized word embedding
D-WISE	Digitale Wissensoziologische Diskursanalyse
DATS	discourse analysis tool suite
DH	Digital Humanities
DTO	Data Transfer Object
DWTS	D-WISE tool suite
<i>e.g.</i>	exempli gratia
<i>etc.</i>	et cetera
FFNN	feedforward neural network
FN	false negatives
FP	false positives
GELU	Gaussian linear unit
GPT	generative pre-trained transformer
GPU	Graphics Processing Unit
GRU	gated recurrent unit
HCDS	Hub of Computing and Data Science
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
<i>i.e.</i>	id est
I/O	input/output
JSON	JavaScript Object Notation
<i>k</i> -NN	<i>k</i> -nearest neighbors
KWIC	Key Word in Context
LISA	Linguistically-Informed Self-Attention
LLM	large language model
LSTM	long short-term memory
LT	Language Technology

mAP	mean average precision
ML	machine learning
MLM	masked language modeling
MTL	multi-task learning
NER	named-entity recognition
NLM	neural language model
NLP	natural language processing
NN	neural network
NTM	neural topic modeling
NUMA	non-uniform memory access
OCR	optical character recognition
PDF	Portable Document Format
PLM	pre-trained language model
PoS	part of speech
QUiTE	QUotes in TExt
ReLU	rectified linear unit
resp.	respectively
REST	REpresentational State Transfer
RNN	recurrent neural network
RSE	research software engineering
SaaS	software as a service
SotA	state of the art
SPA	single-page application
SQL	Structured Query Language
SRL	semantic role labeling
STS	semantic text similarity
T5	Text-to-Text Transfer Transformer
TF-IDF	Term Frequency-Inverse Document Frequency
TFLOPS	Tera Floating Points Operations Per Second
TP	true positives
TPU	Tensor Processing Unit
UI	user interface
UIMA	Unstructured Information Management Architecture
VM	virtual machine
XML	eXtensible Markup Language

List of Figures

1.1	Overview of the parts and chapters and their relations	10
2.1	Generic machine learning processing pipeline applicable to many tasks	12
2.2	Perceptron, the simplest neural network	17
2.3	Activation functions: Sigmoid, tanh, ReLU, and GELU	18
2.4	Three-layer fully-connected feedforward neural network	19
2.5	Two-layer recurrent neural network	24
2.6	Transformer model architecture	26
2.7	Multi-head attention with scaled dot-product attention	27
4.1	Configurable similarity search mockup	50
4.2	Document similarity analysis mockup	50
4.3	Correlation between the dimensions of human judgements	54
4.4	Correlation between the dimensions of the individually finetuned models . . .	54
4.5	Correlation between the dimensions of the multi-task learning model	54
4.6	Correlation differences to human correlations	54
4.7	Overview of the indexing and retrieval process	59
4.8	Mockup of a structured similarity search application	60
4.9	k -NN performance for increasing training dataset sizes for NER	64
5.1	Excerpt from a document annotated with the full relations	79
5.2	Excerpt from a document annotated in the simplified way	79
5.3	<i>Quote</i> token length histograms	82
5.4	Role token length histograms	83
5.5	Number of <i>Quotes</i> per role span	83
5.6	Joint F1 score per foundation model	99
5.7	Joint F1 score per linearization method	100
5.8	Joint F1 score per model size	100
6.1	Visualization of coreference mentions and cluster information	106
6.2	Conceptual visualization of our coarse-to-fine model architecture	111
6.3	Conceptual visualization of our Maverick model architecture	113
6.4	Excerpt from a TüBa-D/Z 10 coreference resolution test set document	122
6.5	Visualization produced by our coreference resolution analysis tool	123
7.1	Annotation interface with annotation scaling suggestions	129
7.2	Quantitative analysis interface	130
7.3	t-SNE entity embeddings from our trained T-NER model	137
8.1	Simple client-server architecture	150
8.2	Client-server communication architecture	151
8.3	OpenAPI workflow	152
8.4	Server high-level architecture	153
8.5	Server scaling architecture	154

8.6	Client architecture	156
9.1	Backend services within API and worker code	164
9.2	Containers of the platform and their interaction	165
9.3	Annotator integrated into DATS	165
9.4	DATS text processing flow	167
9.5	Job processing	168
9.6	Search for similar sentences in DATS	170
9.7	Similarity search result view in DATS	170
9.8	Coreference chains visualization in DATS	172
9.9	Quotation attribution visualization in DATS	173
9.10	Semi-automatic annotation scaling in DATS	176

List of Tables

1.1	Contributions of my publications to the chapters of this dissertation	7
4.1	Pearson correlation with human judgements of our two approaches	52
4.2	mAP scores for retrieving a review of different products but same rating	56
4.3	mAP scores for retrieving a review of same product but different rating	56
4.4	Precision of retrieving a different product with the same rating	57
4.5	Precision of retrieving a review for the same product but different rating	57
4.6	mAP scores for retrieving a narrative of the same category	58
4.7	Named-entity recognition retrieval results	62
4.8	Classification results using k -nearest neighbors	63
5.1	Overview of quotation attribution spans	73
5.2	Overview of the quotation types	74
5.3	Krippendorff’s Alpha agreement between the annotators	80
5.4	<i>Quote</i> media statistics for our annotated dataset	81
5.5	Special tokens used	88
5.6	Quotation detection and attribution dataset splits	93
5.7	Selected evaluation results for quotation detection and attribution	95
5.8	Quotation detection and attribution full development set evaluation results . . .	96
5.9	Quotation detection and attribution full test set evaluation results	97
5.10	Ablation results on the modified dev and test splits	98
5.11	Quotation detection and attribution inference runtime	101
6.1	Overview of the coreference datasets	110
6.2	Performance of coarse-to-fine models with different language models	117
6.3	Performance of coarse-to-fine models with different segment lengths	118
6.4	Performance of our Maverick models with different language models	118
6.5	Main coreference resolution experiment results	120
6.6	Inference compute times of our models	120
7.1	Overview of the datasets used in the annotation scaling experiments	135
7.2	Annotation simulation & k -NN classification results	136

Part I

Introduction, Background & Related Work

Abstract We introduce the challenges of vast textual data in the Digital Humanities (DH) and Computational Social Science (CSS) motivating the application of natural language processing (NLP), and explain the goal of this dissertation to bring state of the art NLP methods for large-scale text collections to DH and CSS. Further, we provide the necessary background on machine learning (ML), neural networks (NNs) and NLP to design suitable methods, and survey general related work of NLP in DH and CSS.

In Chapter 1, we introduce the current challenges that motivate this dissertation, define the research questions, provide an overview of the structure of this dissertation, and specify the contributions with the corresponding published works in the first chapter. In Chapter 2, we explain all required background knowledge referred to throughout this dissertation including basics of ML, NNs and NLP. Finally, in Chapter 3, we present related work relevant to more than a specific chapter by surveying existing research software with NLP assistance, and outlining existing NLP methods for DH and CSS.

1 Introduction

1.1	NLP for Digital Humanities and Computational Social Science	4
1.2	Research Questions	5
1.3	At the Intersection of Social Science, NLP and Software Engineering	5
1.4	Publications	6
1.4.1	Dissertation Related Publications	6
1.4.2	Other Publications	9
1.5	Overview & Content Structure	9

In today’s digital era, ever-increasing amounts of data are produced. This also naturally applies to discourses in natural language — both human-produced and machine-generated. Researchers analyzing such data cannot manually work through the enormous amounts of available data; most likely not even through the amount of relevant data. New data is often produced faster than it can be analyzed even by a large team. Consequently, it is infeasible to only rely on a purely manual analysis. Instead, automatization is essential to deal with large data quantities.

This is also an issue in Digital Humanities (DH) and Computational Social Science (CSS) research where the manual analysis of such large amounts of unstructured text is infeasible (Mcgillivray et al., 2020; Edelmann et al., 2020). For example, when analyzing the effects of climate change on societies, researchers have to process a variety of text types in large quantities such as news articles, non-governmental organization reports, law texts, regulations published by various organizations, international contracts or negotiation documents, social media data *etc.* The sheer volume of the textual data makes it impossible to work through manually, thereby creating a need for automation.

The DH, previously also known as *humanities computing*, are an interdisciplinary area of research connecting the humanities and computer science. In DH, computational or digital methods are applied to humanities research; at the same time humanities methods are applied in the digital world (Warwick et al., 2012). Projects in the field often focus on computing methods applicable to textual materials to either produce large-scale digital text collections or analyze them (Fitzpatrick, 2012). A qualitative analysis typically requires in-depth study of the material that often is accomplished by iteratively reading and manually annotating text passages of interest with codes, selecting the next text(s) to analyze to form an understanding of the researched phenomenon, and finalize the results in a written report (Corbin and Strauss, 2015, pp. 26–34, 82–89). For an overview of the variety of work described as DH, refer to *e.g.* Berry (2012, pp. 1–20) and Schreibman et al. (2004).

CSS is an interdisciplinary field at the intersection of social science and computer science that tests and develops scientific theories about human behavior by applying computational methods on large, often textual datasets from the internet or digitized archives. CSS applies computational methods such as statistical models on complex, large-scale, human behavioral data mostly consisting of textual, location, movement, and network data. Due to the large

data sizes quantitative analyses are performed to answer social science research questions by extracting and aggregating aspects of interest, finding relationships or dependencies within the data, *e.g.* who interacts with whom, how do narratives travel in social networks or how do news outlets differ regarding their presentation of a certain topic. (Cioffi-Revilla, 2010; Edelmänn et al., 2020; Lazer et al., 2020)

Almost all DH and CSS researchers are nowadays confronted with dataset sizes far exceeding manual capabilities. Consequently, researchers started to use automated analysis functionalities to gain an understanding of relevant aspects of those data piles (Mcgillivray et al., 2020). Natural language processing (NLP) approaches have to be applied to many specific use cases and require integration into usable software to make them accessible to the domain experts. NLP is a subfield of computer science that deals with the processing of natural language by a computer, and is nowadays closely connected with machine learning (ML) and artificial intelligence (AI). While there are a number of generic automatizations widely available in common DH or CSS tools such as topic modeling or summarization that can ease some forms of manual analysis by helping to find and explore the different topics in the data, or reducing the data to a more manageable amount. However, other cases require more specific solutions.

Chapter overview In the following sections, we discuss NLP as a way to mitigate data abundance issues in the DH, elaborate on the research questions tackled throughout this work, discuss the challenges of this dissertation arising from the combination of social science, NLP and software engineering, specify the published contributions of this dissertation project linked to the chapters, and finally provide an overview of the structure of this entire dissertation.

1.1 Natural Language Processing for the Digital Humanities and Computational Social Science

In the DH, there is a growing interest to apply NLP methods to explore textual data and scale textual data analysis. The reason for this is twofold. First, due to the advancing digitization of humanities and cultural studies data, both through retro-digitization and the increase in born digital data, large quantities of data are available that are often infeasible for a single person or team to study (Berry, 2012, pp. 1–17). Second, the groundbreaking success of NLP in various disciplines makes it attractive to adapt new methods to the DH domain (Suissa et al., 2022).

Qualitative research requires in-depth study of the material that often is accomplished by manually annotating text. Such endeavors are time-consuming, demanding and, consequently, cannot easily be extended to large-scale corpora. (Creswell and Creswell, 2018, pp. 50–51, 267–272; Corbin and Strauss, 2015, pp. 82–89) The power of state of the art (SotA) NLP models is a great opportunity for qualitative DH researchers to benefit from large datasets where manual in-depth qualitative analysis and annotations cannot be extended to large-scale corpora.

Although NLP has seen groundbreaking success with pre-trained language models (PLMs) and large language models (LLMs), this development has not improved the situation in DH or CSS contexts to the same degree. The black-box behavior of many models, limited applicability, or lacking availability in DH and CSS tooling prevent a widespread adaptation. (Mcgillivray et al., 2020)

1.2 Research Questions

This dissertation deals with the overarching research question that is decomposed into the two more specific research questions:

Overarching Research Question

How can state of the art natural language processing for large-scale text processing be brought to the Digital Humanities and Computational Social Science?

Developing one or more specific NLP methods for each individual DH or CSS use case is infeasible as it does not scale to many use cases. It would likely require one or more NLP scientists and software engineers for each DH resp. CSS researcher to develop and implement the specific method, adapt it as needed, and perform required pre- or post-processing tasks. Thus, the design of generic, widely-applicable NLP methods is to be pursued so that DH and CSS researchers can build upon the results on their own to perform their use case-specific analysis. Consequently, this leads to RQ1:

RQ1: Research Question 1

How can natural language processing methods be designed to be widely applicable to answer many Digital Humanities and Computational Social Science research questions?

While numerous NLP methods might exist that would be helpful for use cases in DH and CSS, they would still not be available to most researchers as methods from NLP research is rarely provided as end-user-facing, ready-to-use software. NLP methods are usually not applicable in the void. They require input data in a certain format, output results in another form and might even require additional services such as a text or vector index to be available. In short, SotA NLP methods need to be integrated into usable software to become available to all DH and CSS researchers. Thus, the second research question RQ2 arises:

RQ2: Research Question 2

How can a wide range of natural language processing methods be made available to Digital Humanities and Computational Social Science practitioners in interactive tooling?

1.3 At the Crossroads of Social Science, Natural Language Processing and Research Software Engineering

This dissertation is set in a challenging environment as this work touches and combines very different areas: Social science, humanities, NLP and research software engineering (RSE). The NLP methods are affected by constraints from DH and CSS and must be usable without programming knowledge. This, in turn, requires software engineering to build usable, robust tools that DH and CSS researchers actually want to use. However, in its core this work deals with NLP methods.

Bringing SotA NLP methods to DH often requires small steps for acceptance in each user group, *e.g.* starting with basic word counts and building up to SotA ML techniques. These methods need to be integrated into easily-usable software that is no throwaway research tool but

a production-grade system. Otherwise, users would never use the software for real projects due to inability to use the tool, risk of data loss, missing basic functionalities or lacking interoperation with existing software *etc.* For CSS, the software has to work well with large data quantities. This requires both suitable NLP methods and an implementation that can actually process the data and allow for quick interactions.

1.4 Publications

This section lists all works that were published as part of the dissertation project in chronological order. Subsequently, my contributions to each collaborative publication are specified. Table 1.1 shows which publication is incorporated into which chapter.

1.4.1 Dissertation Related Publications

- Fynn Schröder**, Hans Ole Hatzel, and Chris Biemann (2021). “Neural End-to-end Coreference Resolution for German in Different Domains”. In: *Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021)*. Ed. by Kilian Evang, Laura Kallmeyer, Rainer Osswald, Jakub Waszczuk, and Torsten Zesch. Düsseldorf, Germany: KONVENS 2021 Organizers, pp. 170–181. URL: <https://aclanthology.org/2021.konvens-1.15/>.
- Steffen Remus, Gregor Wiedemann, Saba Anwar, **Fynn Petersen-Frey**, Seid Muhie Yimam, and Chris Biemann (2022). “More Like This: Semantic Retrieval with Linguistic Information”. In: *Proceedings of the 18th Conference on Natural Language Processing (KONVENS 2022)*. Ed. by Robin Schaefer, Xiaoyu Bai, Manfred Stede, and Torsten Zesch. Potsdam, Germany: KONVENS 2022 Organizers, pp. 156–166. URL: <https://aclanthology.org/2022.konvens-1.19/>.
- Hans Ole Hatzel, **Fynn Petersen-Frey**, Tim Fischer, and Chris Biemann (2023). “Dimensions of Similarity: Towards Interpretable Dimension-Based Text Similarity”. In: *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*. Ed. by Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu. Vol. 372. Frontiers in Artificial Intelligence and Applications. Kraków, Poland: IOS Press, pp. 1004–1011. DOI: 10.3233/FAIA230372.
- Fynn Petersen-Frey** (2023). “Erweiterbare, interaktive Softwareplattform für die Anwendung von Sprachtechnologie in großen Textkorpora zur Unterstützung von Such- und Analyseworkflows in den Digital Humanities”. In: *9. Tagung des Verbands "Digital Humanities im deutschsprachigen Raum" (DHd 2023)*. Ed. by Peer Trilcke, Anna Busch, and Patrick Helling. Trier, Germany & Belval, Luxembourg: DHd 2023 Open Humanities Open Culture. DOI: 10.5281/zenodo.7715444.
- Fynn Petersen-Frey**, Tim Fischer, Florian Schneider, Isabel Eiser, Gertraud Koch, and Chris Biemann (2023). “From Qualitative to Quantitative Research: Semi-Automatic Annotation Scaling in the Digital Humanities”. In: *Proceedings of the 19th Conference on Natural Language Processing (KONVENS 2023)*. Ed. by Munir Georges, Aaricia Herygers, Annemarie Friedrich, and Benjamin Roth. Ingolstadt, Germany: Association for Computational Linguistics, pp. 52–62. URL: <https://aclanthology.org/2023.konvens-main.5/>.

- Florian Schneider, Tim Fischer, **Fynn Petersen-Frey**, Isabel Eiser, Gertraud Koch, and Chris Biemann (2023). “The D-WISE Tool Suite: Multi-Modal Machine-Learning-Powered Tools Supporting and Enhancing Digital Discourse Analysis”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Ed. by Danushka Bollegala, Ruihong Huang, and Alan Ritter. Toronto, Canada: Association for Computational Linguistics, pp. 328–335. DOI: 10.18653/v1/2023.acl-demo.31.
- Fynn Petersen-Frey** and Chris Biemann (2024a). “Dataset of Quotation Attribution in German News Articles”. In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. Torino, Italia: ELRA and ICCL, pp. 4412–4422. URL: <https://aclanthology.org/2024.lrec-main.394>.
- Fynn Petersen-Frey** and Chris Biemann (2024b). “Fine-grained Quotation Detection and Attribution in German News Articles”. In: *Proceedings of the 20th Conference on Natural Language Processing (KONVENS 2024)*. Ed. by Pedro Henrique Luz de Araujo, Andreas Baumann, Dagmar Gromann, Brigitte Krenn, Benjamin Roth, and Michael Wiegand. Vienna, Austria: Association for Computational Linguistics, pp. 196–208. URL: <https://aclanthology.org/2024.konvens-main.22/>.
- Fynn Petersen-Frey**, Hans Ole Hatzel, and Chris Biemann (2025). “Efficient and Effective Coreference Resolution for German”. In: *Proceedings of the 21st Conference on Natural Language Processing (KONVENS 2025): Long and Short Papers*. Ed. by Christian Wartena and Ulrich Heid. Hannover, Germany: HsH Applied Academics, pp. 128–137. URL: <https://aclanthology.org/2025.konvens-1.12/>.

Publication	Chapter							
	3	4	5	6	7	8	9	
Software Platform (Petersen-Frey, 2023)	(✓)					(✓)		
More Like This (Remus et al., 2022)		✓						
Dimensions of Similarity (Hatzel et al., 2023)		✓					(✓)	
Quotation Data. (Petersen-Frey and Biemann, 2024a)			✓					
Quotation Attr. (Petersen-Frey and Biemann, 2024b)			✓				(✓)	
Neural Coreference Res. (Schröder et al., 2021)				✓				
Efficient Coreference Res. (Petersen-Frey et al., 2025)				✓			(✓)	
Annotation Scaling (Petersen-Frey et al., 2023)					✓		(✓)	
The D-WISE Tool Suite (Schneider et al., 2023)						(✓)	✓	

Table 1.1: Contributions of my related publications to the chapters of this dissertation. ✓ denotes a strong contribution, (✓) a partial contribution. Chapters: 3 General Related Work, 4 Semantic Similarity Search for the Digital Humanities, 5 Quotation Detection and Attribution, 6 Transformer-based Coreference Resolution for German, 7 Scaling Manual Annotations to a Large Corpus, 8 Platform, and 9 Applying the NLP methods in the Platform.

For publications with co-authors other than my doctorate supervisor Chris Biemann, my contributions to each paper are specified either by stating which parts other people contributed to my first-author papers or otherwise explicitly naming my contributions to other papers. In publications with Chris Biemann, we jointly developed the research plan, we discussed the general direction, and he contributed a final polish for the camera-ready manuscript while I performed all implementations and experiments, and wrote the vast majority of the paper’s text.

The long paper “Neural End-to-end Coreference Resolution for German in Different Domains” (Schröder et al., 2021) was a collaboration with my colleague Hans Ole Hatzel (equal contribution). He focused on the literary part with the incremental model (Section 3.1, Section 4.2, Section 6). My focus was the news domain and the coarse-to-fine model (Section 3.1, Section 4.0, Section 4.1, Section 5). We jointly worked on the remaining sections of the paper, although we especially contributed the parts most aligned with our focus for the sections on related work and error analysis.

In the long paper “More Like This: Semantic Retrieval with Linguistic Information” (Remus et al., 2022), my primary contributions are in Section 3.0 (Illustration of the indexing and retrieval process), Section 3.2 (Word-level structures), Section 4.1 (Named-Entity Recognition), Section 5 (only named-entity recognition), and Section 6 (only named-entity recognition).

In the long paper “Dimensions of Similarity: Towards Interpretable Dimension-Based Text Similarity” (Hatzel et al., 2023), we jointly worked on Section 1 (Introduction), Section 2 (Background), Section 3 (Methodology), Section 6 (Limitations), Section 7 (Conclusion). My primary contributions are in Section 4.3 (Finetuning), Section 4.4 (Comparing human judgements and machine judgements), Section 5.0 (Down Stream Applications), Section 5.1 (Product Reviews Experiments), and Section 5.2 (Text Register Experiment).

The long paper “From Qualitative to Quantitative Research: Semi-Automatic Annotation Scaling in the Digital Humanities” (Petersen-Frey et al., 2023) is a collaboration with my colleagues Florian Schneider and Tim Fischer, and with project partners Isabel Eiser and Gertrud Koch. As the first author, I contributed the core conceptualization, and wrote most sections. Tim Fischer helped with the example workflow description (Section 3.2), and some experiments on the “Yahoo! Answers” dataset (Section 5). Florian Schneider mainly contributed Section 5.3 that is not part of this dissertation. Isabel Eiser helped with the Introduction (Section 1, especially the motivation), and the related work paragraph on MAXQDA and ATLAS.ti (Section 2).

The demo paper “The D-WISE Tool Suite: Multi-Modal Machine-Learning-Powered Tools Supporting and Enhancing Digital Discourse Analysis” (Schneider et al., 2023a) is a collaboration with my colleagues Florian Schneider and Tim Fischer, and with project partners Isabel Eiser and Gertrud Koch. I contributed to the section on related work (Section 2), the part on annotation in Section 3, and the system architecture (Section 4).

As first author of the short paper “Efficient and Effective Coreference Resolution for German” (Petersen-Frey et al., 2025) I contributed most of the paper. My colleague Hans Ole Hatzel contributed the experiments on literary texts (part of Section 4.2) and their analysis (a part of Section 5).

1.4.2 Other Publications

Fynn Petersen-Frey, Marcus Soll, Louis Kobras, Melf Johannsen, Peter Kling, and Chris Biemann (2022). “Dataset of Student Solutions to Algorithm and Data Structure Programming Assignments”. In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Ed. by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis. Marseille, France: ELRA, pp. 956–962. URL: <https://aclanthology.org/2022.lrec-1.101/>.

Florian Schneider, Seid Muhie Yimam, **Fynn Petersen-Frey**, Gerret Von Nordheim, Katharina Kleinen-von Königslöw, and Chris Biemann (2023). “CodeAnno: Extending WebAnno with Hierarchical Document Level Annotation and Automation”. In: *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Ed. by Danilo Croce and Luca Soldaini. Dubrovnik, Croatia: Association for Computational Linguistics, pp. 11–17. DOI: 10.18653/v1/2023.eacl-demo.2.

Tim Fischer, Florian Schneider, **Fynn Petersen-Frey**, Anja Silvia Mollah Haque, Isabel Eiser, Gertraud Koch, and Chris Biemann (2024). “Extending the Discourse Analysis Tool Suite with Whiteboards for Visual Qualitative Analysis”. In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. Torino, Italia: ELRA and ICCL, pp. 7017–7022. URL: <https://aclanthology.org/2024.lrec-main.615/>.

1.5 Overview & Content Structure

To tackle the research questions, and improve the situation of NLP in DH and CSS, this dissertation has the goal to bring state of the art NLP methods for large-scale text collections to DH and CSS. The dissertation is organized into three parts. Each part begins with a short abstract and brief chapter overview. The chapters begin with an abstract, followed by a listing of the relevant publications (if applicable), and a content overview of the chapter. Figure 1.1 provides a graphical overview over the chapters, their relations, and a minimal preview of their main contents.

In Part I, the work is introduced in context (this chapter), relevant background knowledge is provided on ML, neural networks (NNs) and NLP (Chapter 2), and general related work relevant to the entire dissertation is presented where we review available NLP-assisted research tools and methods for DH and CSS (Chapter 3).

In Part II, we develop novel NLP methods based on neural PLMs and their contextualized word embeddings (CWEs) that are broadly applicable in many DH and CSS contexts with the goal to assist researchers dealing with large text collections answering their specific research questions. Chapter 4 shows two approaches to improve semantic similarity search in DH contexts. We present a configurable similarity search providing insights into which aspects of documents are similar, and a method to search for sub-sentence text spans using semantic search as a modern alternative to KWIC (Key Word in Context). In Chapter 5, we work on automatic quotation detection and attribution. We create a new manually annotated dataset, and develop a novel ML model performing high-quality quotation detection and

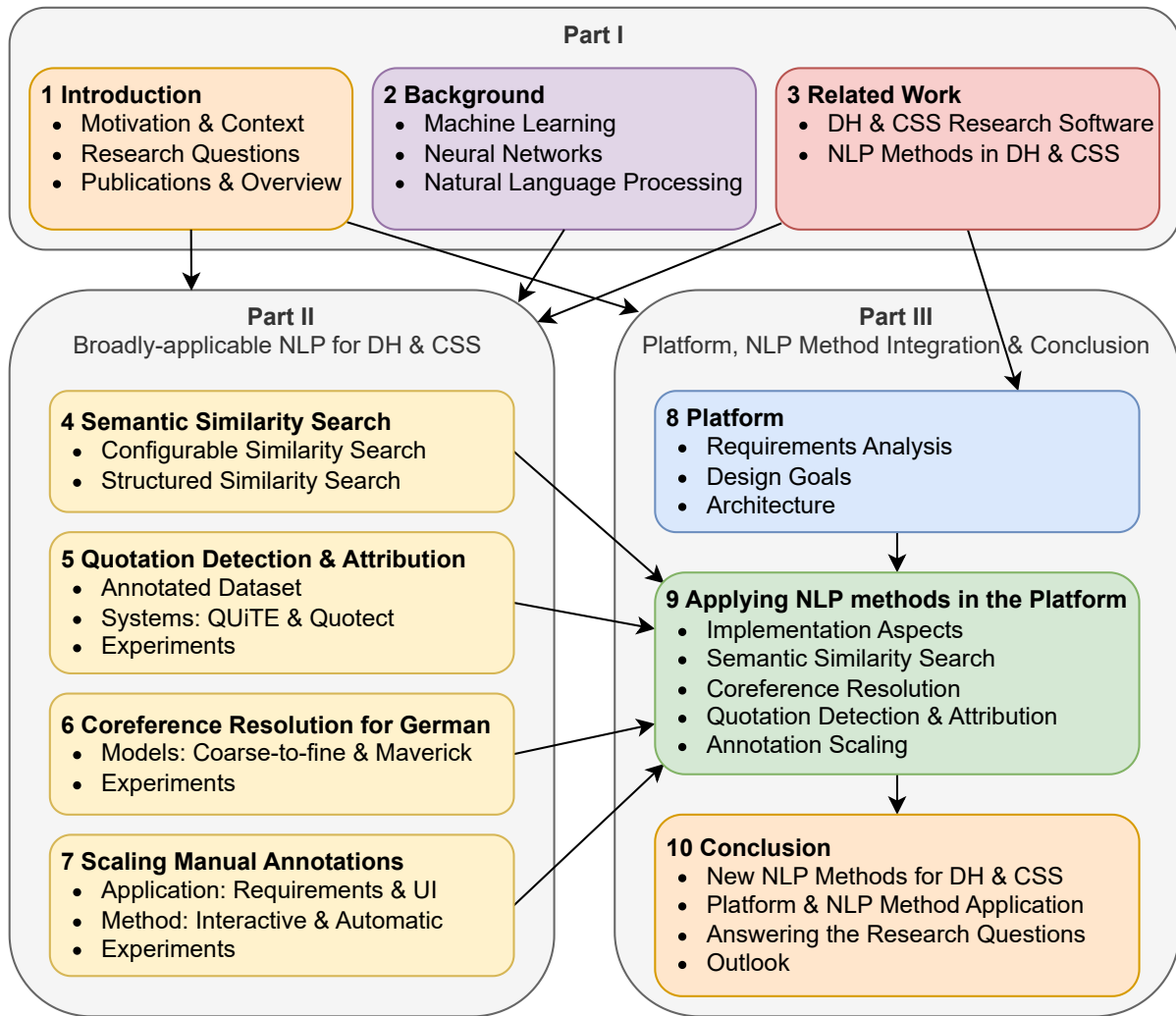


Figure 1.1: Overview of the parts and chapters with their main content. The arrows show the primary dependencies of one chapter on research results or concepts from other chapters.

attribution with vast improvements over other systems. Chapter 6 deals with SotA coreference resolution for German texts. We present two neural models that improve tremendously on previous systems while being computationally efficient even for long documents. In Chapter 7, we present methods to semi-automatically scale manual annotations to large corpora with our envisioned application and interactive method based on CWEs and k -nearest neighbors (k -NN).

In Part III, a suitable open source research platform is conceptually developed (Chapter 8) and implemented into which the methods presented from Part II are integrated (Chapter 9). To create the platform, we perform a requirements analysis to derive the design goals for the platform, and develop suitable architectures for all parts of the platform. We explain key implementation details, and describe the integration of the NLP methods into the platform with all necessary changes. Finally, Chapter 10 concludes this dissertation.

In the next chapter, relevant background knowledge is provided on ML, NNs and NLP.

2 Background

Abstract In this chapter, the necessary background knowledge on machine learning (ML), neural networks (NNs) and natural language processing (NLP) is outlined that allows to follow the development and application of methods described throughout the main parts of this dissertation. It is designed so that it is possible to skip this entire chapter or sections of it, as all explanations in this chapter and their use throughout this work adhere to standard definitions and notation where applicable.

2.1	Machine Learning	11
2.1.1	Fundamentals	12
2.1.2	Evaluation	14
2.2	Neural Networks	16
2.2.1	Basics	16
2.2.2	Feedforward Neural Networks	19
2.2.3	Inference	19
2.2.4	Training	20
2.2.5	Recurrent Neural Networks	23
2.2.6	Gated Recurrent Neural Networks	25
2.2.7	Transformers	26
2.3	Natural Language Processing	28
2.3.1	Text as Symbols	28
2.3.2	Word Embeddings	29
2.3.3	Pre-trained Language Models	29

2.1 Machine Learning

ML is a vast and quickly moving field in computer science revolving around methods that learn from data to solve a particular task without being explicitly programmed how to solve the task. With a substantial increase in computational capacity over the last years, ML has become almost omnipresent as it is being applied in nearly every modality such as text, audio and video across a wide range of domains such as medicine, advertising, games, robotics and social science. Especially tasks which humans solve easily yet classic algorithms are unable to solve adequately are nowadays solved by the means of ML, *e.g.* self-driving cars, translations of text or speech, “artificial intelligence” personal assistants found on every smartphone or even creating art.

As with any powerful tool, unethical misuse is also seen with ML as it is also applied to *e.g.* automatically judge in court, filter applicants in job interviews, track people throughout the digital and physical world against their will and generate fake news texts, images or videos. In this work, ML is used in the context of social science to enable researchers to cope with the large amounts of textual data found on the internet, in news archives or various other sources.

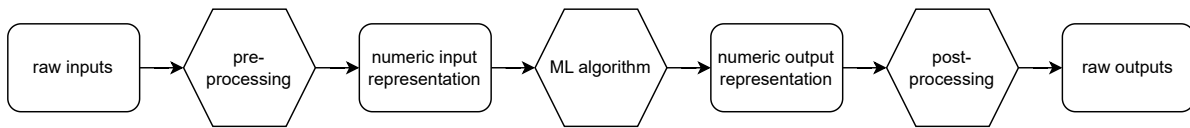


Figure 2.1: Generic ML processing pipeline applicable to many tasks. Raw inputs are pre-processed to numeric input representations which the ML algorithm transforms into another numeric representation that is converted in post-processing to the final usable outputs.

Since a complete introduction and in-depth description of ML in general would go far beyond the scope of this work, please refer to *e.g.* Part I, especially Chapter 5 of the Deep Learning book by Goodfellow et al. (2016) for an extensive description of ML. Thus, this section on ML mainly outlines those aspects required to understand the ML-based NLP methods used and developed throughout this work. As all of these current state-of-the-art NLP methods are based on NNs, the focus is set on neural ML. Before NNs will be outlined beginning with Section 2.2, the fundamentals of classic ML will be briefly described in the upcoming section.

2.1.1 Fundamentals

The overall idea of ML is to feed data into an algorithm instead of explicitly modeling a solution. Typically, the problem is too difficult to develop a fixed program manually. Instead, a domain expert provides relevant data points and a suitable ML algorithm is applied on these examples with the goal to either detect structures in the data in an unsupervised manner or to reproduce the expert’s solution in a supervised setting and generalize it to unseen data. In its essence, ML is pattern recognition with statistical methods. The “learning” in machine learning refers to the learning from the provided data to produce outputs.

In the upcoming subsections, typical ML tasks are introduced followed by the necessary training (Section 2.1.1.2) and evaluation (Section 2.1.2) of ML algorithms.

2.1.1.1 Tasks

The task or objective describes how the ML system processes a certain input and what it should produce as an output. Raw inputs to the ML system can be almost anything like a digital image, a text or a graph representation. The ML algorithm operates on numerical values, typically a vector $\mathbf{x} \in \mathbb{R}^n$ with fixed or dynamic length of n . Consequently, a ML system includes some pre-processing to transform the raw input into a numerical representation suitable for the algorithm. As an example, a digital image might be resized to a fixed size and the RGB pixel values are fed into the algorithm as matrix. Analogously to the input, the output sometimes also needs to be transformed. The post-processing step converts the numerical output produced by the ML algorithm back into a commonly understood representation such as a digital image or text. For trivial cases such as a score between 0 and 1, the post-processing function is just the identity function. Figure 2.1 visualizes the above description as a generalized ML pipeline.

While a wide variety of very specific tasks can be solved by ML systems, they are often variants of one of the following common, generic tasks (see also Goodfellow et al. (2016, pp. 97–101)):

Classification The classification task requires an algorithm to decide which of k previously defined classes an input belongs to. The task can be modeled in different ways. A ML system could produce a single numeric value directly representing a class, *e.g.* the number 5 corresponds to class 5 out of k classes in total. More useful is the output of all possible classes together with their confidence scores. In this case, the ML system produces a vector $\mathbf{x} \in \mathbb{R}^k$ and at later step the highest confidence value is chosen with the index in the vector specifying the class. For some cases a probability distribution over the classes is required, thus the output of the vector $\mathbf{x}$ is constrained with $\sum_{i=0}^k x_i = 1$. Typical examples of a classification task could be object recognition in an image, hate-speech detection in online comments.

Regression In a regression task, the ML algorithm is asked to produce a single numerical value for an input example. Thus, the algorithm should behave as function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. As the task is similar to classification (except the output format), a simple classification can be modelled as a regression task and vice versa. An example regression task could be scoring product reviews or movie plots. Such a regression task could also be formulated as a classification task with *e.g.* 5 classes (1–5 stars as typically encountered in product reviews).

Transcription In a transcription task, the ML system converts one rather unstructured type of data into another, usually one input is transcribed into a sequence of discrete text symbols. This transcription converts between different modalities such as from image to text (*e.g.* optical character recognition (OCR)) or speech to text (speech recognition).

Translation The translation task requires a ML system to convert one sequence of symbols in a language into a new sequence of symbols of another language. As an example, a sequence of *e.g.* English words is converted to a sequence of German words. While translation is most commonly performed on textual data, speech-to-speech translation between different languages is also applied in some applications.

Synthesis In a synthesis task, the ML system has to produce new examples similar to those of the training inputs. This is helpful when the creation a large variety of similar items is required, *e.g.* texture generation for video games. A special case is synthesis given a specific input, *e.g.* performing speech synthesis to generate a natural sound for the given text.

2.1.1.2 Training

In contrast to classic algorithms where all required knowledge is part of the algorithm itself, ML algorithms require training on data to learn how to solve their task. ML algorithms can roughly be divided into supervised and unsupervised learning depending on the type of information provided to them during their training (see also Goodfellow et al. (2016, pp. 102–105)).

Unsupervised Machine Learning In unsupervised learning, the algorithm is provided a dataset containing many features, and it learns helpful properties of structure of this dataset. This can be an approximation of the probability distribution that generated the dataset or a way to cluster the dataset into groups of similar examples. In NLP, the language modeling is likely the most common unsupervised ML: Given a large dataset of plain text, the ML models learns the probability distribution of word sequences by predicting the next word or filling masked words.

Supervised Machine Learning In supervised learning, the algorithm is not only provided with the raw data during training but also a so-called label or target value for each input example. The algorithm is asked to predict the correct label for each input. While these labels are typically created manually by humans, there are also many scenarios where labels are generated automatically, *e.g.* where one computationally expensive system generates labels to train a more efficient ML system. Most NLP tasks are currently solved by supervised ML systems where humans provide(d) the correct labels during training, *e.g.* named-entity recognition (NER), spam classification, sentiment analysis, sentence similarity.

2.1.1.3 Example: k -Nearest Neighbors

One of the simplest supervised learning algorithms that still sees plenty of practical applications is k -nearest neighbors (k -NN). It is actually a group of algorithms that can be used for both classification and regression. Since they are non-parametric learning algorithms, there are no trainable parameters at all. Thus, there is no training or learning phase required. Instead, during inference k -NN predicts the output of each test input by considering the k -nearest data points in the labeled training data. k is a so-called hyperparameter, specifically an integer value selected by *e.g.* trial and error during the development phase or more elaborate means such as automatic hyperparameter optimization. Depending on task and data, a suitable metric to define the distance between any pair of data points is needed, *e.g.* to compute the distance between two points in a two-dimensional space the Euclidean distance can be used. For k -nearest neighbor classification, the algorithm first finds the k -nearest data points of a test input within the labeled training data. Then, it selects the majority value of the labels of the k examples. While classification works with almost any label, *e.g.* numbers, colors, boolean values *etc.*, k -nearest neighbor regression only supports number as it returns the average of the k -nearest labels. (Hastie et al., 2009, pp. 14–15, 463–475)

2.1.2 Evaluation

ML models are evaluated to assess their quality. In general, evaluation is performed on a held-out part of the dataset that was not used at all during training of the model. For supervised ML, the models predictions are evaluated against human-labeled data points, often referred to as *gold* data. Depending on the task, a suitable evaluation metric is used to compare the predictions with the gold data.

Supervised Classification One of the simplest measures for classification tasks is *accuracy*:

$$\text{accuracy} = \frac{\text{correct classifications}}{\text{all classifications}} \quad (2.1)$$

Due to its flaws with imbalanced label/class distribution, the more robust F_1 score, the harmonic mean of precision and recall, is typically used instead Goodfellow et al. (2016, pp. 417–419):

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (2.2)$$

Precision and recall can be defined using the number of true positives (TP), false positives (FP) and false negatives (FN):

$$\text{precision} = \frac{TP}{TP + FP} \quad (2.3)$$

$$\text{recall} = \frac{TP}{TP + FN} \quad (2.4)$$

In words, precision is the number of true positive predictions divided by the number of all inputs predicted as positive (regardless of their actual label). Recall is the number of true positive predictions divided by the number of all inputs that actually are positive. While precision, recall and F_1 score are by far the most frequent classification metrics, some tasks require more specialized metrics, *e.g.* the task of coreference resolution (see Chapter 6, page 105).

Supervised Retrieval Evaluation measures for information retrieval systems assess the quality of the results a search system returns from a typically large document collection in response to the user's query. The standard precision, recall and F_1 score metrics can also be used in a retrieval scenario (Carterette and Voorhees, 2011):

$$\text{precision} = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{retrieved documents}|} \quad (2.5)$$

$$\text{recall} = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{relevant documents}|} \quad (2.6)$$

However, precision and recall do not take the order of the results into account. Average precision is a specialized retrieval measure that also considers the order of the results. It computes the average value of precision at an interval of recall between 0 and 1,

$$\text{average precision} = \sum_{k=1}^n P(k) \Delta r(k) \quad (2.7)$$

where $P(k)$ is the precision at cut-off k in the result list, or simply precision at k , and $\Delta r(k)$ is the change in recall from items $k - 1$ to k in the returned list. Mean average precision (mAP) is the mean of the average precision of a series of n queries.

$$\text{MAP} = \frac{\sum_{q=1}^n \text{average precision}(q)}{n} \quad (2.8)$$

For large, web-scale retrieval, recall becomes a less useful metric because it requires knowledge of all relevant documents (practically impossible for the entire internet) and many queries have far more relevant documents than the user wants to read. In such cases, precision at k ($P@k$) still remains a useful, simple metric, *e.g.* $P@5$ corresponds to the number of relevant documents among the top five retrieved documents. (Carterette and Voorhees, 2011)

Unsupervised Evaluation of unsupervised ML largely depends on the goal why an unsupervised method is employed. There are metrics to directly evaluate the unsupervised performance, *e.g.* clustering comparison measures to evaluate the similarity between two

clusterings (see *e.g.* Palacio-Niño and Berzal (2019) for an overview). Often, the unsupervised learning is part of a larger model that is later evaluated with labels in a supervised manner. Another form of evaluation that arguably mistakenly is sometimes seen as an unsupervised evaluation is actually supervised evaluation for models that are trained in an unsupervised manner, *e.g.* perplexity to evaluate masked language modeling (MLM) on a model that was trained to fill artificially masked gaps in text.

2.2 Neural Networks

In an artificial NN many simple units called neurons work together through their vast connections between them to solve a potentially complex task. A NN with enough neurons which are suitably connected and use a fitting activation function can approximate any continuous function (Hornik et al., 1989). In theory, NNs are able to solve any complex task that can be formulated using numeric input and output. In practice however, it is often a challenge to find the right architecture and connection weights to solve a specific task well.

This section contains both a short introduction to simple, general NNs and more advanced models typically used in NLP. For a complete introduction and in-depth explanation of NNs, refer to *e.g.* Part II, especially chapters 6–10 of the Deep Learning book by Goodfellow et al. (2016).

2.2.1 Basics

In this section, the very basics of NNs are briefly outlined: The concept of an artificial neuron, the Perceptron (Rosenblatt, 1957) as the simplest NN, and activation functions that are applied on the neurons. With these building blocks larger and more powerful feedforward neural networks (FFNNs) can be created as shown in the next section.

2.2.1.1 A Single Artificial Neuron

As introduced above, a neuron is a simple unit and the basic building block of any artificial NN. There are three types of neurons. Input neurons receive their inputs from external sources outside the NN. These are the numeric input representations produced from the raw inputs (see Figure 2.1). Hidden neurons receive their inputs from other neurons and provide their output to other neurons. Output neurons provide the final outputs to the external world in form of the numeric output representation that usually needs to be post-processed into externally usable outputs.

Each input to a single neuron is individually weighted (in practice: multiplied) by the weight of the connection. The individually changeable weights form the learnable parameters of a neuron. By adjusting the weights of all neurons a NN can be trained to solve a specific task. (Schmidhuber, 2015)

The basic functioning principle of a neuron is to sum all of its weighted inputs, *i.e.* to compute a weighted sum. On this sum, it applies a so-called activation function and outputs its result. Unless it is an output neuron, this result is then used as the input for other neurons. By connecting to other neurons, multiple neurons form a complex NN. The key component of any activation function is that it introduces a non-linearity into the whole computation of the network allowing to solve non-linear tasks — the majority of real-world tasks. (Freund and Schapire, 1999)

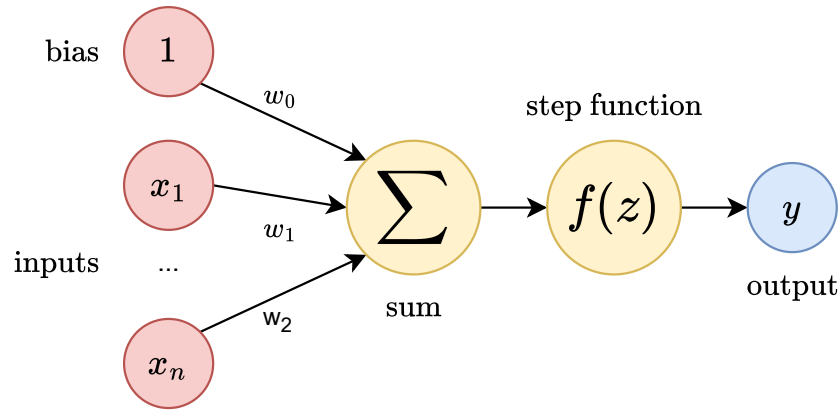


Figure 2.2: Perceptron, the simplest NN. The inputs (red) are combined with a weighted sum (yellow) and the step function (yellow) determines the final output value (blue).

2.2.1.2 Perceptron

The perceptron by Rosenblatt (1957) is a simplistic NN with only a single neuron. It acts as a binary classifier producing either *false*: 0 or *true*: 1 as output from a real-valued input vector that is mapped to multiple inputs of the single neuron. Figure 2.2 depicts a graphical representation of the perceptron's components. The single neuron acts as both an input and output neuron at the same time. The perceptron uses the step function as an activation function. It outputs 1 if the weighted sum is positive, 0 otherwise. In addition to the normal inputs, the perceptron includes one constant input called bias. The bias has a value of 1 and a learnable weight w_0 to add any value to the weighted sum allowing to shift the activation function to the left or right. Thereby, the perceptron can output 1 also when the weighted sum without the bias would be negative. This enables to produce a different output for two input examples producing weighted sums of identical sign, e.g. two positive numbers. (Freund and Schapire, 1999)

As mentioned in the introduction of this section, the challenge with NNs is to find the correct weights to solve a certain task. For a perceptron, the training to adjust its weights is simple. Given a training example in form of a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ where x_i is the value of the i th input feature, c as the correct output value for this training example, the learnable weight vector $\mathbf{w} = (w_0, w_1, w_2, \dots, w_n)$ initialized randomly, the $f(z)$ as activation function and λ as a learning rate, two steps are performed iteratively to adjust $\mathbf{w}$ to train the perceptron to solve a task: (1) Calculate the perceptron's output $y = f(\sum \mathbf{w} \cdot \mathbf{x}) = f(w_0x_0 + w_1x_1 + \dots + w_nx_n)$ (2) Update the perceptron's weights $0 \leq i \leq n$: $w_i = w_i + \lambda \cdot (c - y)x_i$. Multiple training examples are required to enable learning robust values of $\mathbf{w}$ generalizing to many inputs. Training can be stopped after e.g. a fix amount of iterations or once a specified training error threshold is reached. To solve tasks more complex than linearly separable problems (Minsky, 1987) multiple neurons need to be connected together to form a proper NN as shown in Section 2.2.2.

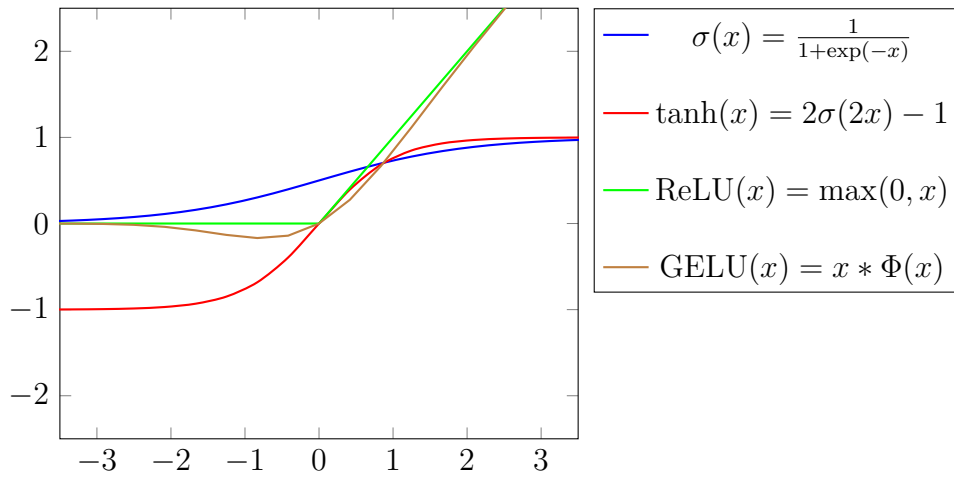


Figure 2.3: Activation functions: Sigmoid, tanh, ReLU, and GELU

2.2.1.3 Activation Functions

Activation functions introduce a non-linearity into the whole network of neurons to be able to solve non-linear tasks. While the perceptron used the step function, current NNs use other activation functions with additional desirable properties such as producing a strong gradients or derivatives (more on that in Section 2.2.4), and computational efficiency (Ramachandran et al., 2017). Figure 2.3 shows four common activation functions: sigmoid, tanh, rectified linear unit (ReLU) and Gaussian linear unit (GELU).

The sigmoid function $\sigma(x) = 1/(1+\exp(-x))$ transforms any real-valued input to a positive number between 0 and 1. In NNs, it is typically not used in hidden neurons due to its low gradients but frequently employed at the output in case a task requires a valid output range of 0 to 1 (Goodfellow et al., 2016, pp. 178–180).

The hyperbolic tangent function $\tanh(x) = 2\sigma(2x) - 1$ squashes any real-valued input to a number between -1 and $+1$. It features two valuable properties compared to the sigmoid function: It avoids bias in the gradients and provides stronger derivatives (Glorot and Bengio, 2010), especially in the common input range of -1 to $+1$.

The Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$ simply clips a real-valued input to be at least 0 or greater. Although it is not differentiable at $x = 0$, this poses no problems for implementations, and it provides a constant strong gradient of 1 for all positive inputs while being very fast to compute (Goodfellow et al., 2016, pp. 168–171). Providing a strong gradient is important for training NNs with many neurons in succession (Krizhevsky et al., 2012).

The Gaussian Error Linear Unit (GELU) $f(x) = x * \Phi(x)$ where $\Phi(x)$ is the cumulative distribution function for a Gaussian distribution. It is a fully differentiable alternative to the ReLU function delivering strong gradients for positive inputs. However, it is more expensive to compute. (Hendrycks and Gimpel, 2023)

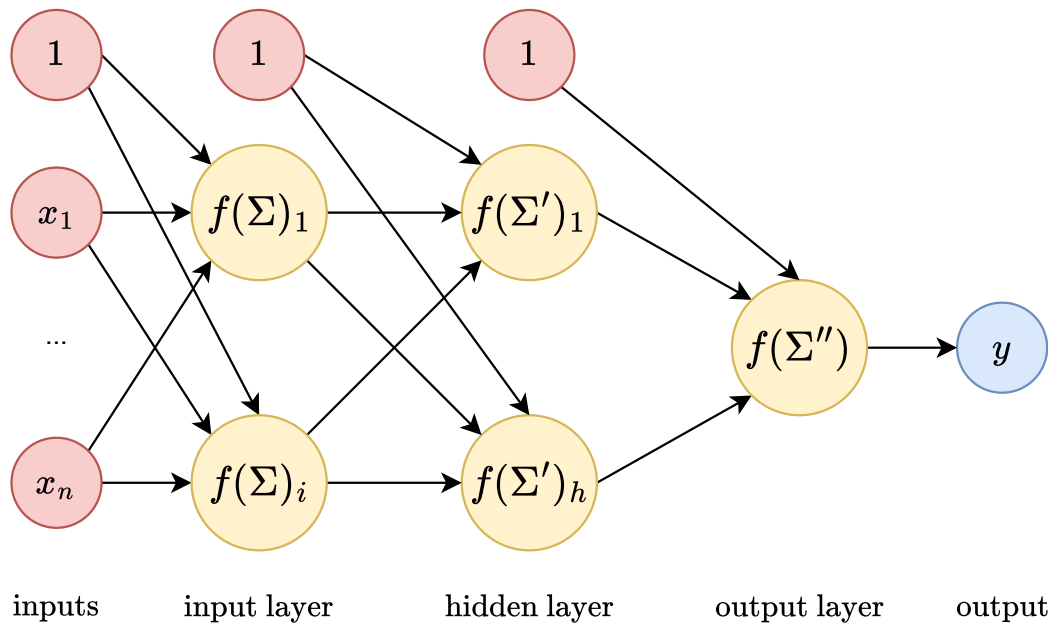


Figure 2.4: Three-layer fully-connected feedforward neural network with n input features, i input neurons, h hidden neurons and one output neuron.

2.2.2 Feedforward Neural Networks

Since a single neuron cannot solve nonlinear problems (Minsky, 1987), more neurons are needed to solve complex tasks. A FFNN is shown in Figure 2.4. FFNNs always have exactly one input and output layer with any number of hidden layers in between. Each layer can consist of an arbitrary number of neurons. FFNNs grow in depth by adding more hidden layers and/or width by adding more neuron per layer. The name *feedforward* describes that each neuron feeds its output into the next layer. Thus, in a FFNN there are no connections between neurons within the same layer or back to any previous layer. Typically, the outputs of layer j are the inputs of layer $j + 1$. However, there are network architectures with skip-connections where e.g. some layer j feeds its outputs to layer $j + 1$ and/or $j + 2$. (Goodfellow et al., 2016, pp. 164–167)

FFNNs with multiple output neurons become multi-class classifiers instead of a purely binary classifier possible with a single output. With enough neurons and a non-linear activation function FFNNs become universal approximators that can approximate any measurable function to any degree of precision given the correct weights (Hornik et al., 1989).

2.2.3 Inference

Inference describes the process of performing all computations required to produce the NN's output from a given input. For a network with multiple layers such as feedforward architecture, the outputs of each layer need to be computed sequentially as the output from layer i is needed as input of layer $i + 1$. The computations for each neuron within a layer can be performed in parallel as long as neurons have no connections within the same layer or to previous layers — which is the case for a FFNN. For each neuron a number of operations are computed

sequentially as shown in Figure 2.2: The weighted sum of the inputs is computed first, followed by the activation function. (Goodfellow et al., 2016, pp. 164–172)

In the following, the operations are described that are performed during inference on the three-layer network shown in Figure 2.4. As every input is connected to each neuron in the input layer with an individual weight, the i th input for the j th input neuron is multiplied with the weight w_{ij} belonging to the connection. For each input neuron, the weighted inputs are summed. For fast processing, the weights are stored as a two-dimensional matrix $\mathbf{W}$ with n columns and m rows, and the inputs are stored as single vector $\mathbf{x}$ of length n . Then, all multiplications and summations required in a single layer can be performed highly efficiently with a single matrix-vector product $\mathbf{W}\mathbf{x}$ to produce an m -dimensional vector $\mathbf{h}$ where the j th component contains the weighted input sum for input neuron j . Now, the activation function can be computed in parallel as an element-wise operation on vector $\mathbf{h}$ resulting in a new vector $\mathbf{h}'$. The processing for the hidden layer is performed analogously to the input layer: Instead of $\mathbf{x}$, $\mathbf{h}'$ contains the inputs and since the weights are different another matrix $\mathbf{W}'$ contains the weights of the hidden layer. Additional layers would be processed analogously with new weight matrices using the outputs of the previous layers as inputs. Computing the final network's output is again a matrix-vector product followed by the activation function of the output layer. As the network in Figure 2.4 only has a single output, the result is a vector $\mathbf{y}$ of length 1.

Inference in FFNNs is a highly parallelizable and fast to compute as long as the network is much wider than deep, *i.e.* has few layers with many neurons each. Due to the parallelizable nature, computations are nowadays usually performed on GPUs (Graphics Processing Units) or TPUs (Tensor Processing Units) that excel at performing many simple computations with high degree of parallelism compared to classical CPUs (Central Processing Units) that are more tailored towards sequential processing. For network architectures other than FFNNs, inference might involve additional steps *e.g.* to handle circular connections. Before dealing with other architectures, the process of training a NN will be explained in the next section.

2.2.4 Training

While inference is both easy to understand and fast to compute, training a NN with multiple layers is complex and significantly slower to compute. As described in Section 2.2.1.2, the weights of a single neuron can be directly adjusted from actual and desired outputs in an iterative manner. In order to train multi-layer NNs, more elaborate techniques called *error backpropagation* with gradient-based learning are required (Rumelhart et al., 1986). The term *backpropagation* refers to propagating the error from the output back through all the network's layers to the first layer.

For a detailed description of backpropagation with gradient-based learning, refer to *e.g.* Goodfellow et al. (2016, 172pp) for gradient-based learning and Goodfellow et al. (2016, 200pp) for backpropagation. In the following, the basic principle will be outlined.

Given a NN that computes a function f during inference, the idea of gradient-based learning is to minimize a task-specific loss function $L(\mathbf{x}, \mathbf{y}^*, \theta)$ with input $\mathbf{x}$, the correct output $\mathbf{y}^*$ and the network's weights that form all its learnable parameters θ . Gradient descent is an algorithm that iteratively minimizes the loss as computed with L by adjusting the network's weights in the direction of the steepest descent of the loss. To apply gradient-based learning

in multi-layer networks, backpropagation is used to efficiently compute the direction of the steepest descent for each individual weight. To obtain the gradient of the loss function, the derivative of f is required which can be calculated using the chain rule¹ of every function (matrix multiplications and activation functions) applied during the forward pass.

With every input, three steps are performed iteratively to train a network: (1) Calculate the network's output $\mathbf{y} = f(\mathbf{x})$ (2) Compute the error gradient e of loss function L for every weight using backpropagation (3) Modify the network's weights $\mathbf{w} = \mathbf{w} + e \cdot \lambda$. With non-linear activation functions, training a FFNN is a non-convex optimization problem. Thus, there is no guarantee of finding a global minimum. In practice, this is usually not an issue as gradient descent in the high-dimensional space can find a local optimum that is close the global optimum (LeCun et al., 2015).

With inference and training solved for FFNN, new challenges await when processing sequences of variable length. Since the number of inputs is fixed, variable-length inputs such as text pose a challenge. Given a fixed number of inputs, shorter inputs need to be artificially prolonged with special empty inputs, so-called *padding*, and longer inputs would need to be sliced into small-enough chunks and processed individually, thereby losing the ability to process a whole document at once. Such issues with sequences can be solved with the help of recurrent neural networks (RNNs) (see Section 2.2.5).

2.2.4.1 Regularization

Regularization is a crucial concept in training large NNs with many parameters to avoid overfitting. When a network overfits, the training error is low while the error on test data is high. This occurs because large networks with many parameters tend to memorize the entire training data including noise by learning a too complex solution instead of generalizing. Numerous regularization techniques have been proposed to reduce overfitting and improve generalization, *i.e.* parameter norm penalties (L^1 , L^2), data augmentation, parameter sharing, early stopping and dropout. The different techniques can be combined for a stronger effect. (Goodfellow et al., 2016, pp. 224–225)

Parameter norm penalties Parameter norm penalties add a regularization term $\Omega(\theta)$ to the loss function L to introduce a penalty on the network's weights. The regularized loss function $\tilde{L}$ is defined as $\tilde{L}(\theta; \mathbf{x}, \mathbf{y}) = L(\theta; \mathbf{x}, \mathbf{y}) + \alpha\Omega(\theta)$, where α is a real-valued hyperparameter that controls the influence of the regularization term $\Omega(\theta)$ to the original loss function L . The regularization term $\Omega(\theta)$ only depends on the parameters θ independent of the network's input and output. It is usually only applied to the weights and excludes the biases since bias values are essential to shift the activation function according to the problem. L^2 regularization defined as $\Omega(\theta) = \frac{1}{2} \|\theta\|_2^2$ is called weight decay because it drives the weights closer to zero. L^1 regularization is defined as $\Omega(\theta) = \|\theta\|_1 = \sum_i |\theta_i|$. It results in sparser solutions than L^2 regularization with many parameters being 0, thereby acting as a network-wide feature selection mechanism. (Goodfellow et al., 2016, pp. 226–233)

Data augmentation Arguably the best way to prevent overfitting and improve generalization in NNs is to use more training data. With data augmentation artificial data is created by making tiny changes to the original training samples that do not affect the expected output.

¹https://en.wikipedia.org/wiki/Chain_rule

While the modification of some data like image (shifting, rotation, changing brightness) or sound (changing pitch or speed) is easy, many other data types such as text cannot be safely augmented. (Goodfellow et al., 2016, pp. 236–238)

Parameter sharing With parameter sharing some weights either within a single network or between multiple networks are tied together. Tying related parameters ω_1 and ω_2 is achieved by applying a parameter norm penalty on their difference: $\Omega(\theta) = \frac{1}{2} \|\omega_1 - \omega_2\|_2^2$. When reusing the same network parameter for multiple inputs as in a convolutional neural network (CNN) processing image data an even stronger form of parameter sharing is achieved. Instead of using unique weights for each input pixel, the same set of weights is reused for each pixel, thereby increasing the number of data points used to train a single weight. (Goodfellow et al., 2016, pp. 249–251)

Early stopping While the training error decreases over time with more epochs, the validation error often increases again with too many training epochs because the model starts to overfit the training data. Early stopping is a simple technique to prevent this overfitting by stopping the training before the validation error increases. Thus, the validation error is computed after each epoch. Whenever a lower validation error is obtained, the current network weights are saved. The entire training is stopped, and the best weights are reloaded once the validation error has not improved for a few epochs. Early stopping is straightforward to implement, not constrained to specific network types, and actually reduces the required amount of compute because training typically stops before the maximum number of epochs. In theory, the effect of early stopping is similar to L^2 regularization with model weights initialized close to zero because it restricts the weights to stay close to their initial values. (Goodfellow et al., 2016, pp. 241–249)

Dropout Dropout (Srivastava et al., 2014) performs noise injection at hidden units to approximate an ensemble of similar networks. It works for all types of networks and can be combined with other regularization techniques. During training, hidden neurons are randomly deactivated by multiplying their outputs with zero. In each batch, all hidden neurons are independently deactivated with a probability p . This creates individual “subnetworks” with some neurons missing while all remaining weights are shared. During inference, all neurons are active with their weights rescaled by p , so the overall output is similar to the training phase. Dropout forces the network to learn similar, redundant neurons or adjust the neurons in the next layer to infer the missing information. (Goodfellow et al., 2016, pp. 255–265)

2.2.4.2 Hyperparameter optimization

A NN not only consists of learnable weights, it also requires suitably setting numerous adjustable architecture or training parameters for a network to solve its designated task effectively and efficiently. Hyperparameters, such as the number of layers, type of neurons, activation functions, learning rate, regularization settings *etc.*, greatly affect the performance of a model and its computational costs. While manual tuning of hyperparameters requires a deep understanding of NNs and experience, automatic selection reduces these knowledge requirements by increasing computational cost orders of magnitude. (Goodfellow et al., 2016, pp. 422–431)

Hyperparameter selection is closely connected to the relation between training and test error, and between model capacity and training data size. Model capacity describes how much information a model can store or how complex the functions it can approximate can be. A model with higher capacity requires more training data to not overfit. The effect of many hyperparameters can be explained with model capacity. Increasing the number of hidden units increases the model capacity as well as required compute resources. Selecting a higher dropout probability decreases model capacity as it becomes challenging for neurons to fit the training data. Similarly, increasing the amount of parameter regularization reduces model capacity as it restricts the possible values of weights. While the learning rate does not affect the theoretical capacity, it influences the effective capacity as minimizing the loss function fails with a too high or too low learning rate. (Goodfellow et al., 2016, pp. 422–426)

Automatic hyperparameter tuning works by finding the parameters achieving the best results on a validation set. In contrast to early stopping comparing the scores every epoch, hyperparameter selection considers the final score after each entire training run, which itself already is an expensive operation. Thus, it is computationally highly expensive. While hyperparameters could be tested from all possible values, usually a fixed set or range of values is supplied by the user or framework. For very few hyperparameters trying all combinations as produced with their Cartesian product via plain brute-force is a typical solution called grid search. The possible values per hyperparameter are often selected from a logarithmic scale to cover a wide range. For best results, grid search needs to be repeated narrowing down the possible values. (Goodfellow et al., 2016, pp. 427–429)

A more efficient alternative to grid search is random search (Bergstra and Bengio, 2012). Instead of computing all combinations while changing only a single hyperparameter at a time, random search chooses all hyperparameters randomly from a range or distribution of possible values. Although random search converges faster to good values, it still requires repeated runs with a refined range of possible values. (Goodfellow et al., 2016, pp. 429–430)

2.2.5 Recurrent Neural Networks

Although FFNNs are efficient and effective for applicable tasks, they lack proper sequence processing capabilities. Numerous tasks, especially in the field of NLP, require dealing with sequences. Sequences can be used as input, output or both. Document classification is a standard task with a sequence as input and a fixed-size output. Generating a sequence as output from a fixed-size input could be the task of producing a textual description of an image. The classic example for variable-length sequence as both input and output, called sequence-to-sequence, is machine translation (Sutskever et al., 2014; Cho et al., 2014).

RNNs are specialized in processing sequences of variable length for both inputs and outputs such as text or audio. The network computes a recurrent function $\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta)$ for each time step t with $\mathbf{x}^{(t)}$ as the input at time step t , $\mathbf{h}^{(t)}$ the representation of the hidden layer at time step t , and θ as the network's weights. The hidden representation $\mathbf{h}^{(t)}$ is fed as input to the output layer for each time step t of the sequence to generate the actual output. Compared to a FFNN with large amounts of inputs, RNNs also allow parameter sharing, *i.e.* using the same weights for different positions of a sequence, thereby leading to increased generalization capabilities and fewer required training examples (Karpathy, 2015). An example RNN with a single hidden layer is shown in Figure 2.5. In case of multiple hidden layers, the outputs

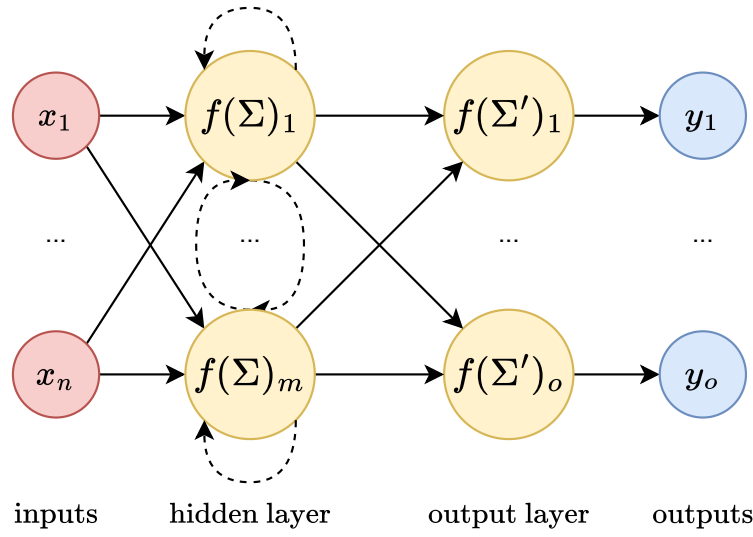


Figure 2.5: Two-layer RNN with n input features, m hidden and o output neurons. Recurrent connections are dashed, and the bias is omitted for clearer display.

from the last hidden layer would feed back into the first hidden layer as recurrent connections. (Goodfellow et al., 2016, pp. 367–368)

2.2.5.1 Inference

To perform inference in a RNN, the recurrent function is converted into a linear chain of repeated operations for each sequence individually as the function f is repeated n times for a sequence of length n . Unfolding the above recurrent definition yields a chained application of f while reusing the identical network weights θ :

$$\mathbf{h}^{(t)} = f(f(f(\dots f(\mathbf{h}^{(0)}, \mathbf{x}^{(1)}; \theta) \dots, \mathbf{x}^{(t-2)}; \theta), \mathbf{x}^{(t-1)}; \theta), \mathbf{x}^{(t)}; \theta)$$

where $\mathbf{h}^{(0)}$ is the initial hidden state filled with zeros or random numbers close to zero. (Goodfellow et al., 2016, pp. 369–372)

To perform the forward computations during inference, a RNN as shown in Figure 2.5 requires three operations for each time step t using the weight matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ for input-to-hidden, hidden-to-output and hidden-to-hidden connections (Goodfellow et al., 2016, pp. 372–376):

- (1) Summing up the inputs $\mathbf{x}^{(t)}$ together with the recurrent connections:

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$$
- (2) applying the activation function f to produce the new hidden state:

$$\mathbf{h}^{(t)} = f(\mathbf{a}^{(t)})$$
- (3) feeding the hidden state into the output layer to produce the outputs $\mathbf{y}$:

$$\mathbf{y}^{(t)} = \mathbf{V}\mathbf{h}^{(t)}$$

As each time step depends on the result of the previous computations, the process is inherently sequential and cannot be parallelized. Only the operations within a single hidden layer are parallelizable as in a FFNN. Consequently, the inference runtime increases linearly with the sequence length.

2.2.5.2 Training

Training a RNN is similar to training a FFNN with gradient-based learning and backpropagation using the unfolded computations. More specific, the backpropagation through time (BPTT) algorithm is used to account for the shared parameters of recurrent connections. To compute the gradients and propagate errors from the loss function at the output layer back through all chained computations the hidden states of all time steps need to be kept in memory. Thus, the backward pass has linear space and time requirements. Since a RNN has to be trained with many examples over multiple iterations, this becomes a computationally expensive process in comparison to a FFNN. (Goodfellow et al., 2016, pp. 374–380)

While training a RNN is expensive, a much bigger issue are the *vanishing gradient* and *exploding gradient* problems first described by Bengio et al. (1994). The multiplication of many numbers between 0 and 1 exponentially shrinks to 0, and multiplying many numbers greater than 1 exponentially goes to infinity. The matrix multiplications in the hidden layer of a RNN exhibit this very issue leading to exponential gradients with longer sequences. In case of vanishing gradients, a network's weights are most changed by the last time steps of a sequence. For exploding gradients, the first time steps of a sequence far outweigh the later steps. Both cases result in failure to learn temporal correlations between distant positions in a sequence (Pascanu et al., 2013). A solution to these problems are gated RNNs.

2.2.6 Gated Recurrent Neural Networks

Gated RNNs were developed to solve the training issues of plain RNNs. The goal of gated RNNs is to achieve computations with connections through time that feature constant gradients eliminating the exploding/vanishing gradient problem. The gated RNNs first proposed is the long short-term memory (LSTM) developed by Hochreiter and Schmidhuber (1997). It features *cells* containing a self-loop and weights to control the weight change of the self-loop connection. Long-term information is stored as internal state per *cell* without being changed by gradient-based updates. Instead, the gated RNNs learns weights in its *gates* via gradient-based updates to control when and how much to update the current, internal state with new information. When combining many of these cells together, a LSTM network is created using LSTM cells instead of simple neurons in the hidden layers. The cells are still recurrently connected. (Goodfellow et al., 2016, pp. 404–407)

Since the original LSTM more variants have been developed to improve certain aspects. Gers and Schmidhuber (2000) add *peephole connections* enabling the gate layers to look at the hidden state thereby conditioning the self-loop weights on the context to improve performance. Cho et al. (2014) developed the gated recurrent unit (GRU) as a minimalist variant that replaces forget- and input-gate with a single update-gate, and combines cell and hidden state making the GRU simpler and faster to compute.

Gated RNNs have successfully been applied in many sequence processing tasks such as speech recognition (Li and Wu, 2015), machine translation (Sutskever et al., 2014) or parsing (Kiperwasser and Goldberg, 2016). Although gated RNNs overcame the training issues of plain RNNs and achieved new state-of-the-art results, they are slow to compute (Culurciello, 2018) as the sequential computations cannot be parallelized and are limited by memory bandwidth resulting in inefficient implementations. Thus, both training and inference of gated RNNs is even more time-consuming than with plain RNNs. (Goodfellow et al., 2016, pp. 407–408)

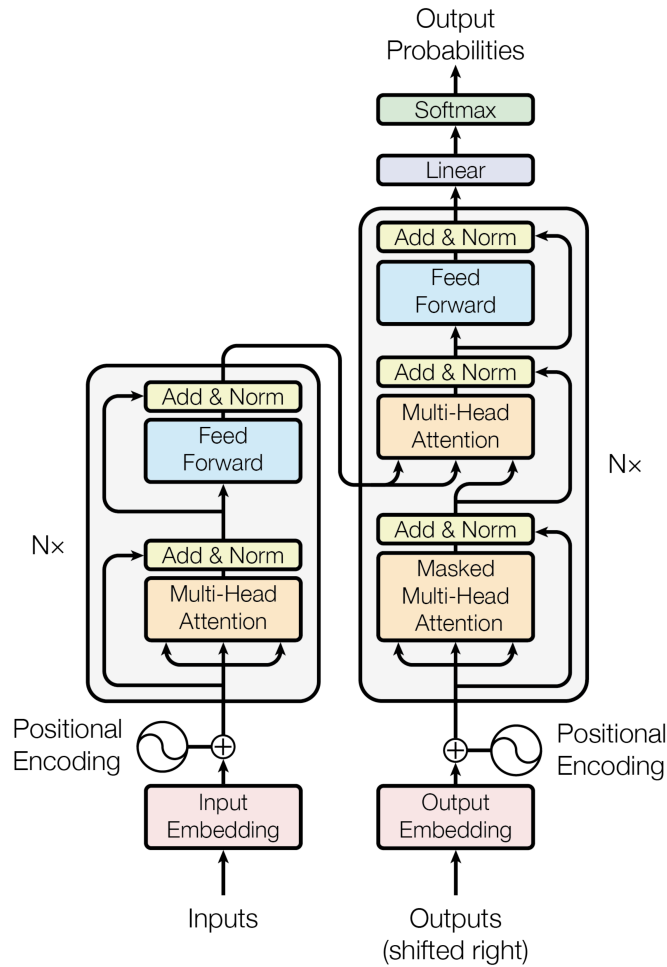


Figure 2.6: Transformer model architecture, taken from Vaswani et al. (2017)

2.2.7 Transformers

The biggest issue with (gated) RNNs is that their training cannot be parallelized, thereby making it infeasible to scale such networks to millions or even billions of parameters and large training corpora. The Transformer architecture (Vaswani et al., 2017) has revolutionized the field of NNs improving on the previous state of the art in many tasks. It is solely based on an attention mechanism without any recurrence or convolutions yet capable of processing long sequences. The main advantages over previous architectures are to draw global dependencies between input and output, and enable a large degree of parallelism allowing it to scale well to billions of parameters and large corpora.

The Transformer architecture is an encoder-decoder structure allowing to process sequences and produce an output sequence of arbitrary length. Figure 2.6 shows the model architecture. While the encoder maps an input sequence of symbols to a sequence of vector representations, the decoder generates a sequence of symbols from both the current input and previous output representation, thereby making it autoregressive. For both encoder and decoder, it uses stacked self-attention, fully connected FFNNs and learned embeddings to convert the input and output tokens to vector representations. As the Transformer has no recurrence, a positional encoding

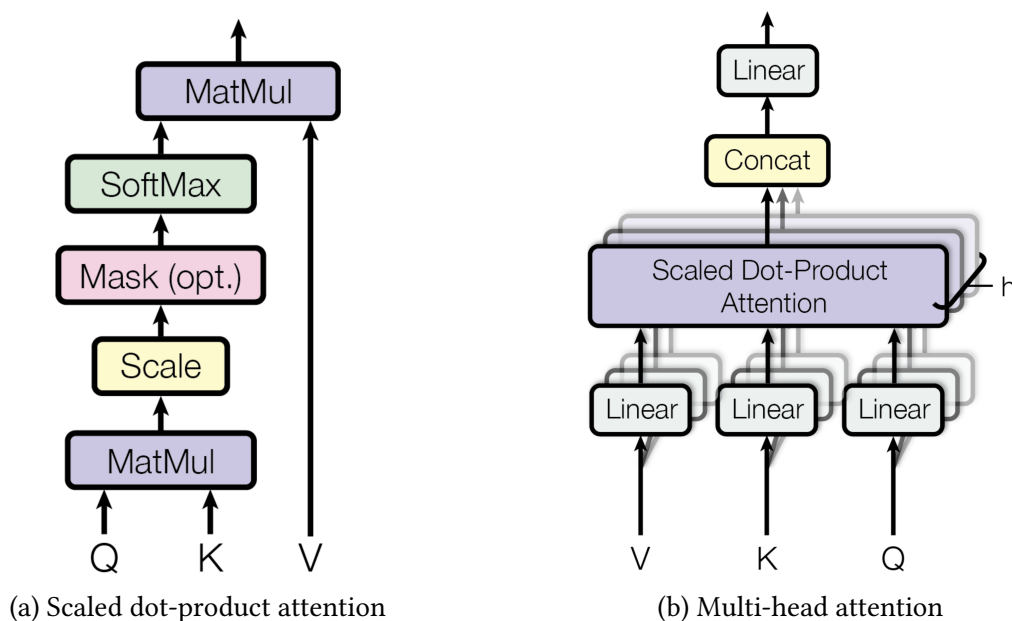


Figure 2.7: Multi-head attention running several attention layers with scaled dot-product attention in parallel. Images taken from Vaswani et al. (2017)

is integrated into the embeddings to inject information regarding the position of tokens in the sequence. (Vaswani et al., 2017)

Encoder and decoder stacks Both encoder and decoder are stacked $N = 6$ times with multiple sub-layers having residual connections (He et al., 2016) followed by layer normalization (Ba et al., 2016). The encoder has a multi-head self-attention and a fully connected FFNN as sub-layers. The decoder has a third multi-head self-attention sub-layer integrating the encoder output. (Vaswani et al., 2017)

Attention mechanism The attention mechanism is a function mapping a query and key-value pairs to a weighted output with all components being vectors. The weights of each value are computed by the compatibility between query and key. The scaled dot-product attention shown in Figure 2.7a computes the dot products of the query with all keys, scales by dividing with the number of key dimensions, and applies a softmax function to finally weight the values. Multi-head attention performs h scaled dot-product attentions in parallel on smaller linear projections of the original key, value and query vectors. This allows the model to jointly attend to information from different representation subspaces at different positions. The self-attention mechanism has quadratic complexity with the sequence length, making it faster than RNNs for short sequences like single sentences. (Vaswani et al., 2017)

Efficient long sequence processing The full attention with quadratic complexity becomes slow and infeasible to compute for long sequences such as entire documents. Numerous works have tackled the issue of creating a faster Transformer. The Longformer (Beltagy et al., 2020) combines a local sliding window attention with a limited global attention to process longer sequences efficiently. Wang et al. (2020) propose the Linformer with a new self-attention mechanism based on low-rank matrix approximation, which reduces the overall self-attention

to linear complexity. Katharopoulos et al. (2020) express the self-attention as a linear dot-product of kernel feature maps and make use of the associativity property of matrix products to also achieve linear complexity. Choromanski et al. (2021) propose Performers which can estimate full attention Transformers with provable accuracy using only linear complexity. While these approximate approaches reduce the theoretical complexity, they incur some quality degradation without being much faster in practice due to hardware limitations and implementation issues. FlashAttention (Dao et al., 2022) and its successors (Dao, 2023; Shah et al., 2024) take a different approach by computing the full attention with implementations optimized for modern GPUs by fusing operations together, reducing accesses to relatively slow memory, and enabling better parallelism. In practice, modern large-scale Transformer networks are relying on FlashAttention for training and inference.

2.3 Natural Language Processing

NLP is the art and science of enabling computers to consume, understand and produce human language. It allows computers to process human language in order to find patterns, reveal the underlying structure, understand the meaning, extract information, answer questions, make a conversation *etc.* In this section, the most important foundations of modern NLP in the era of NNs are explained: Text as symbols, word embeddings and pre-trained large language models (LLMs). For a more complete overview refer to *e.g.* Jurafsky and Martin (2026).

2.3.1 Text as Symbols

For a NN to process text, words need to be represented as a numeric vector. In a first step, (sub-)words or single characters are converted into symbols in the form of an integer value, its index in a vocabulary. The ASCII (American Standard Code for Information Interchange) uses 1 byte to represent each character resulting in a tiny vocabulary of 256 entries when directly feeding text as a stream of characters into a NN. While feasible, this approach is challenging for a NN as needs to learn the concept of words on its own during training. Further, this requires the most compute resources as each individual character is processed by the network.

In contrast, converting all individual words into symbols produces a large vocabulary, thereby making it challenging for the network to learn rare words. Moreover, the vocabulary would need to be of infinite size as an infinite number of words can be constructed from arbitrary character sequences. As an infinite vocabulary cannot be used for computations, a common approach is to select the n most frequent words and replace everything else with a special *unknown* symbol for out of vocabulary.

Since both approaches have severe downsides, the solution used in current neural NLP is to convert sub-words, or pieces of words into symbols with WordPiece (Devlin et al., 2019) tokenization or modified Byte-pair encoding (Gage, 1994). Similar to general-purpose compression algorithms, text is encoded into symbols referring to variable-length character sequences that are most frequent. Thus, common words are encoded as one symbol while rare words are split into two or more sub-word pieces. This combines the best of both worlds: The vocabulary size becomes configurable, each text sequence can be represented without loss of information and processing is fast as frequent words only require a single symbol.

2.3.2 Word Embeddings

NNs cannot meaningfully operate on the symbols stored as integer values directly. Instead, they need to be converted into a vector representation. The symbols or integers can be translated into a one-hot vector, where all positions are 0 except the index of the symbol to be represented. Such one-hot vectors are still far from optimal for a NN as most inputs are 0 all the time as NNs operate on dense vectors of floating point values. Word embeddings are the solution to transform words via symbols into dense representations that work well with NNs. (Jurafsky and Martin, 2026, pp. 132–137) Each word vector represents a position in the high-dimensional embedding space. Such representations are not only useful for processing in NNs. By computing *e.g.* the Euclidean distance or the cosine similarity between different embeddings, similar words can be identified, *i.e.* words that are semantically related. (Jurafsky and Martin, 2026, pp. 96–105)

The concept of word embeddings is based on the distributional hypothesis (Harris, 1954) stating that words in similar contexts have similar meanings. A breakthrough for practical application came with the efficient embedding algorithm *word2vec* proposed by Mikolov et al. (2013) who trained a skip-gram model with the negative-sampling method. The skip-gram model is trained on plain text to predict the surrounding context (previous and text words) given a single input word.

*fastText*² (Joulin et al., 2017) is an even faster method to train and use word embeddings. While *word2vec* operates on whole words (with *unknown* symbols), *fastText* employs sub-word information improving the performance in downstream tasks especially for unseen words (Bojanowski et al., 2017).

Both *word2vec*, *fastText* and other early, nowadays less popular techniques are static word embeddings as they generate the same representation for a word irrespective of its context. *ELMo* (Peters et al., 2018) was the first method to create contextualized word embeddings (CWEs). A character-based multi-layer bidirectional LSTM network produces a unique numeric vector for each word depending on its context. This enabled to model polysemy, *i.e.* words with the same surface form having different meanings.

Devlin et al. (2019) introduce BERT (Bidirectional Encoder Representations from Transformers) based on the Transformer architecture by Vaswani et al. (2017). Among other things, BERT creates context-dependent embeddings like *ELMo* using sub-word symbols instead of characters. In summary, the CWEs originate from NN models that use encoded (sub-)words as symbolic input. A CWE is a unique vector representation of a (sub-)word depending on the entire text processed by the NN in one forward pass.

2.3.3 Pre-trained Language Models

Following the NN revolution with the Transformer architecture (Vaswani et al., 2017), pre-trained language models (PLMs) based on this architecture such as generative pre-trained transformer (GPT) (Radford et al., 2018), BERT (Devlin et al., 2019) and T5 (Text-to-Text Transfer Transformer) (Raffel et al., 2020) continued the transformation of the field of NLP and achieved a new state of the art on numerous NLP tasks. The PLMs resulted in a paradigm shift in NLP. Before, a single model was trained on a labeled dataset for each specific task

²<https://fasttext.cc/>

from scratch. While word embeddings helped, each model still needed to first learn latent feature representations of language in order to understand language and then learn to solve the actual task. All non-trivial NLP tasks require a certain understanding of language which is challenging for a NN given the typically small labeled training datasets. (Min et al., 2023)

With PLMs a foundation model is trained on huge amounts of unlabeled text data with a highly generic pre-training task to learn a general language understanding. GPT is based on Transformer decoder blocks and is trained with next token prediction to get the probability distribution of a word given all previous words, thereby conditioning the model on the previously generated tokens in a unidirectional manner. BERT is build on Transformer encoder blocks and primarily relies on MLM where a small percentage of tokens is masked, and the model is trained to replace the mask with the correct token, thereby conditioning the model on the entire context in a bidirectional manner. T5 is a standard encoder-decoder Transformer pre-trained with a text-to-text denoising objective inspired by MLM on large unlabeled text, and then fine-tuned on numerous tasks reformulated into text-to-text format.

According to Min et al. (2023), three paradigms have emerged with using PLMs. In *pre-train then fine-tune* a PLM such as BERT is taken as a foundation providing general language understanding and fine-tuned for a specific task by either training a new classifier on top the representations of its last layers (CWEs) or by adding additional output layers and jointly training the new and pre-trained model weights.

With *prompt-based learning* a generative model such as GPT is not trained by changing any model weights. Instead, an instruction and a few training examples (“few-shot learning”) are provided as input context (often in form of input-output pairs following a common template) together with the actual text to be processed with the instruction.

Last, *NLP as text generation* deals with converting non-generative tasks such as NER into text generation problems. This allows a single model such as T5 to solve numerous NLP tasks with a unified framework using the same model weights. In contrast to prompt-based learning, the PLM is fine-tuned for one or more specific tasks by adapting its weights using labeled training data.

Since these initial Transformer-based PLMs, numerous improved variants have been proposed. RoBERTa (Liu et al., 2019) keeps the BERT architecture but optimizes the pre-training data and methodology. Conneau et al. (2020) introduce XLM-RoBERTa (XLM-R) increasing model size and training data over the initial multilingual BERT for improved performance across many benchmarks. Clark et al. (2020) developed ELECTRA training Transformer encoders as discriminators by corrupting the original sequence with a replaced token detection pre-training task. It learns with fewer samples, thereby achieving performance on par with RoBERTa with greatly reduced training compute power.

For generative PLMs, typically referred to as LLMs, the model and training data size has skyrocketed within only a few years, e.g. the prominent GPT series went from 117 million parameters to 1.5 billion with GPT-2 (Radford et al., 2019), 175 billion with GPT-3 (Brown et al., 2020) and an estimated 1.8 trillion with GPT-4 (OpenAI et al., 2024). Their increase in performance and ability to solve more tasks in few- or even zero-shot settings come at the price of tremendous amount of compute and data required, thereby making the creation of such large models only possible for a handful of companies in the world, and leaving universities or other public organizations far behind.

3 General Related Work

Abstract In this chapter, general related work is described that applies to this dissertation as a whole or to at least to two or more different chapters. Specific related work can be found in the “Related Work” sections in each chapter in Part II of this dissertation. First, we survey existing natural language processing (NLP)-assisted research software including NLP annotation tools, Digital Humanities (DH) and Computational Social Science (CSS) analysis tools, and active learning (AL) tools. Subsequently, we discuss existing NLP methods for DH and CSS that are either commonly available in tools or specifically implemented for each project.

This chapter is (partly) based on the following publication(s):

Fynn Petersen-Frey (2023). “Erweiterbare, interaktive Softwareplattform für die Anwendung von Sprachtechnologie in großen Textkorpora zur Unterstützung von Such- und Analyseworkflows in den Digital Humanities”. In: *9. Tagung des Verbands "Digital Humanities im deutschsprachigen Raum" (DHd 2023)*. Ed. by Peer Trilcke, Anna Busch, and Patrick Helling. Trier, Germany & Belval, Luxembourg: DHd 2023 Open Humanities Open Culture. DOI: 10.5281/zenodo.7715444.

3.1	NLP-assisted Research Software in DH and CSS	31
3.1.1	NLP Annotation Tools for Linguistic Annotations	31
3.1.2	DH and CSS Annotation and Analysis Tools	33
3.2	NLP Methods in the Digital Humanities and Computational Social Science . .	37
3.2.1	Data Exploration	38
3.2.2	Search	39
3.2.3	Extracting Structured Information	40

3.1 NLP-assisted Research Software in Digital Humanities and Computational Social Science

Numerous applications have been developed to annotate or analyze text documents. However, there are no peer-reviewed encompassing surveys known to the author. Toscano et al. (2022) only review software solutions for web information systems in DH. We will discuss relevant available software in this section regarding their functionality, integrated NLP methods, and software architecture.

3.1.1 NLP Annotation Tools for Linguistic Annotations

Many existing NLP annotation tools are designed for the *manual* creation of linguistic annotations with the goal to perform a fine-grained annotation on every document of a fixed corpus.

brat¹ (Stenetorp et al., 2012): Open-source, web-based application to add fine-grained structured annotation to plain text. It is specifically designed for linguistic annotations and frequently used to create data for NLP tasks. It is the foundation for the pure annotation part found in WebAnno and Inception.

WebAnno² (Yimam et al., 2013, 2014; Eckart de Castilho et al., 2016): Open-source, complex annotation tool with many additional features over brat such as user management, inter-annotator agreement calculation, curation, import/export, different roles, progress monitoring *etc.* It is the first software of a family of web-based collaborative platforms for linguistic annotations based on the Apache UIMA (Unstructured Information Management Architecture) framework³. While it can be configured to support almost any kind of annotation including relations and span chains, it lacks support for multipart spans. It includes an automatic method where the system learns from user-provided annotations. However, this requires expert machine learning (ML) knowledge to perform feature engineering and training. WebAnno is designed for linguistic annotations, not to perform a qualitative content analysis, and it does not support large-scale corpora.

Inception⁴ (Klie et al., 2018): An open-source successor of WebAnno build on the same technical basis (Java, UIMA), with a brat-based annotator, enhanced with a more modern user interface (UI), and lots of additional features including knowledge base support and built-in ML classifier training. It includes a AL mode to guide annotators to improve the system by labeling examples providing valuable information to the classifier. Like WebAnno, Inception is primarily designed for linguistic annotations, not for qualitative content analysis, and it does not support large-scale corpora.

CodeAnno (Schneider et al., 2023b): An open-source extension of WebAnno that features document-level hierarchical annotations and automations using user-trained ML models.

TextAnnotator (Abrami et al., 2020): A closed-source, web-based application for the simultaneous and collaborative annotation of text. Similar to the applications WebAnno, Inception and CodeAnno, it allows multiple users to annotate a shared document collection with multiple distinct types of linguistic annotation, provides inter-annotator agreement calculation and a knowledge base. It uses a client-server architecture with external tools for pre-processing, authorization *etc.*, and Apache UIMA framework for annotation storage.

LabelStudio⁵ (Tkachenko et al., 2020) is an open-source annotation platform that offers AL and ML functionalities requiring set-up by connecting external models, thereby making it less suitable for domain experts. However, it provides not only NLP functionalities but also computer vision, speech processing, time series and more. It is a client-server application with a modern web-based client, and a server written in Python.

¹<https://brat.nlplab.org>

²<https://webanno.github.io/webanno/>

³<https://uima.apache.org>

⁴<https://inception-project.github.io>

⁵<https://labelstud.io>

Summary The tools reviewed in this section are designed to manually create a labeled corpus. This annotated corpus is then typically used to train and/or evaluate a ML classifier. While it is perfectly possible to export the annotated documents solely for a direct analysis, this is by no means the intended usage as the export formats are not easily human-readable. Further, the tools do not include state of the art (SotA) technology or automatic preprocessing for easier or (semi)automatic annotations. These tools (except CodeAnno) are not made for coarse-grained, *i.e.* document-level, annotations. Moreover, all these tools lack an overview of the document collection and general search functionality, making it practically infeasible to apply them for projects that are not pure annotation but require an iterative process of data selection, annotation, annotation schema refinement *etc.*

3.1.2 DH and CSS Annotation and Analysis Tools

In the following, we discuss the features and limitations of popular (open-source) computer assisted qualitative data analysis software (CAQDAS), general purpose annotation tools and analysis software concerning quantitative analysis for DH and CSS on large document collections.

Tools that are widely used in DH and CSS are the two proprietary software tools MaxQDA⁶ and Atlas.ti⁷ that come with all the common issues of proprietary software such as requiring expensive licenses which often poses a hurdle of academic research groups. Moreover, due to being closed-source research groups cannot adapt the software for their specific projects needs if required. Further, the tools do not support specialized, advanced deep learning features. As an alternative to these two big proprietary products, a number software packages or services are available which we will discuss in the following paragraphs.

For quite specific tasks, there are numerous tools designed such as:

1. SiNLP (Crossley et al., 2014): Tools tailored for discourse analysis using dictionaries and word count-based statistics to be used by researchers inexperienced in NLP with a focus on simple surface-level features of words in a text. Single-user desktop software.
2. ALCIDE (Moretti et al., 2014) short for “Analysis of Language and Content In a Digital Environment”: Analysis tool tailored for historical and political discourse. It is a closed-source, web-based tool providing automatic text pre-processing and provides features such as a geographical distribution from metadata, named-entity recognition (NER), keyword extraction, text-based search.
3. new/s/leak (Yimam et al., 2016; Wiedemann et al., 2018) supports investigative journalists in finding interesting facts and connections in large textual data leaks. It features a graph view of named entities and keywords with their relations, frequency information, document timeline, and search.
4. LawStats (Ruppert et al., 2018): Web-based tool for search and analysis of court decisions from the Bundesgerichtshof (Federal Court of Justice). It provides a classifier for the revision outcome together with quantitative insights via a faceted search.
5. Network of the day⁸ (Benikova et al., 2014) is an open-source web-based application for interactive visualization of social media and online news articles. It automatically

⁶<https://maxqda.com>

⁷<https://atlasti.com>

⁸<http://tagesnetzwerk.de>

extracts the most important entities (such as persons or organizations) discussed and their relations for each individual day, and shows them in an interactive network. User can tag the relations between entities, look at the occurrences of most popular entities over time, and search for specific entities to track developments in the media over time. The main issue with these tools is that they are rather poorly suitable to work on other issues than their specific single use-case.

Recogito⁹ (Simon et al., 2017) is an intuitive, web-based open-source tool for qualitative and quantitative research with a focus on places and the integration in a map via automatic detection of entities. While it also includes annotation and collaborations functionalities, Recogito misses a powerful search functionality and lacks support for large corpora. It follows standard a client-server architecture with Java-based server using external services for database (PostgreSQL) and text-search index (ElasticSearch). Recogito is not maintained anymore as the latest release is from 2019.

CATMA¹⁰ (Gius et al., 2022) is an intuitive, web-based open-source tool for qualitative and quantitative research with a focus on annotation including excellent support for multipart span annotations. However, it lacks support for relations between different spans. While it allows configurable annotations schemata and multiple analysis functions for annotations and the plain text, CATMA misses a powerful search functionality, does not provide meaningful automatic pre-processing, and lacks support for large corpora. CATMA is Java-based server application using a web browser to display its UI. It stores the user data versioned in git repositories and is actively maintained as of 2025.

Voyant Tools¹¹ (Sinclair and Rockwell, 2016) is an open-source web-based application for text reading and analysis with a focus on supporting interpretive practices for DH students and scholars. It features tools to importing documents in various formats, several tools for studying word or term frequencies and distributions within document(s) or the entire corpus, a full-text reader that support very large texts and includes interactive features, interaction between the tools that facilitates navigation and exploration at different scales (from "close reading" to "distant reading"), visualize the frequency and distribution of words and terms in a corpus or document, a collocation graph for keywords and terms, KWIC (Key Word in Context), correlations between term frequencies, document/corpus term frequencies, common phrases, scatter plot how words cluster in a corpus using document similarity, correspondence analysis or principal component analysis, and general corpus statistics. Voyant follows a classic client-server software architecture, with the web application as a client communicating with the single, monolithic server. It is actively maintained as of 2025.

Open Semantic Search¹² (Mandalka, 2023) is an open-source, web-based application for searching, exploring and text mining of heterogeneous, large document collections. It features a document processing pipeline with crawling, text extraction, NER recognition, and optical character recognition (OCR) for images and PDF (Portable Document Format). Its search interface allows full-text search, faceted search, exploratory search and knowledge graph

⁹<https://recogito.pelagios.org/>

¹⁰<https://catma.de>

¹¹<https://voyant-tools.org>

¹²<https://opensemanticsearch.org>

search. Further, it provides means for document tagging and annotation by integrating the proprietary hypothesis¹³ platform. For analysis, it has an aggregated overview of named entities, visualization of locations in an interactive map, a graph view for relations between named entities via co-occurrences, trend chart visualization, word clouds, and document comparisons via frequencies of overrepresented concepts or entities. It has a complex software architecture with numerous external services enabling all its functionalities, and is distributed as fully configured virtual machine (VM) or as numerous Linux containers via docker compose for container management. Apache Solr¹⁴ is used for search, Apache Tika¹⁵ for content extraction, spaCy (Honnibal et al., 2020) for NER, Celery¹⁶ with RabbitMQ¹⁷ for task management, and neo4j¹⁸ as its graph database. Besides these external services, it consists of a worker service that handles the pipeline processing, and a Django¹⁹-based web applications to serve the UI and a REST (REpresentational State Transfer) API (Application Programming Interface).

Annotation Studio²⁰ (Folsom et al., 2015) is an open-source collaborative web application for the annotation of texts with notes with a focus on technology-supported pedagogy. It does not require prior experience with close textual analysis or TEI (Text Encoding Initiative). Its features are mainly focused on education in a teacher-student setup.

AntConc²¹ is a closed-source single-user desktop freeware software for corpus and text analysis providing nine different tools (Anthony, 2024). The KWIC tool shows search results in a typical KWIC format to see how the search term is used in the corpus. A plot tool shows in which places of each document a search term occurs. A file tool simply shows the full text of a single document for reading. The cluster tool presents the most common contiguous multi-word patterns for a single-word search term. The *n*-gram tool shows the most frequent *n*-grams for a user-selected value of *n*. A collocate tool finds words that frequently appear near to a given search term to find co-occurring words in the corpus. A word list tool lists all words in the entire corpus sorted by their number of occurrences. The keyword tool lists words that are unusually frequent the analyzed corpus in comparison to a reference corpus. The wordcloud tool visualized the results of the previous tools as a word cloud. Last, the chat tool integrates large language models (LLMs) to interact with the LLM using results generated by the other tools. Since AntConc is closed-source, the internal software architecture is unknown. As it is a single-user desktop software without a server, there is no form of collaboration.

CorpusExplorer²² (Rüdiger, 2018) is an open-source desktop software for corpus linguistics and text mining providing dozens of analysis functionalities with many automations. It features support for plenty of import/export file formats, a web crawler, integration of taggers for part of speech (PoS) and syntactic trees, manual annotations, similar document finder,

¹³<https://hypothes.is>

¹⁴<https://solr.apache.org>

¹⁵<https://tika.apache.org>

¹⁶<https://docs.celeryq.dev>

¹⁷<https://www.rabbitmq.com>

¹⁸<https://neo4j.com>

¹⁹<https://www.djangoproject.com>

²⁰<https://www.annotationstudio.org>

²¹<https://www.laurenceanthony.net/software/antconc/>

²²<https://notes.jan-oliver-ruediger.de/software/corpusexplorer-overview/>

limited distributed processing, support for add-ons providing additional functionality, and numerous analysis functions such as: frequency analyses, phrases and n -grams, phrases, co-occurrences, disambiguation, sentiment detection, KWIC, style metrics, corpus distributions. Since CorpusExplorer is a single-user desktop software without a server, there is no form of collaboration.

Mallet²³ (MAchine Learning for Language Toolkit) is an open-source command line program providing statistical NLP and ML-based NLP for document classification, clustering, topic modeling, and information extraction (McCallum, 2002). As Mallet is a single-user program without a graphical UI, it is less user-friendly than the previous software packages but still usable without programming knowledge.

WebLicht²⁴ (Hinrichs et al., 2010) is a web-based application integrating a multitude of rather old NLP modules for linguistic tasks that can be used to build individual processing pipelines in a drag-and-drop fashion. It provides individual web services for tokenization, sentence splitting, PoS tagging, lemmatization, parsing, chunking, word-sense disambiguation, and NER. Thus, it uses a service-oriented software architecture with a central database storing metadata about each available service, a web-based frontend to access the central service coordinating the decentralized processing services. WebLicht is not designed for collaboration or interactive analysis. Instead, the main goal is to enrich input texts with the results of the processing pipeline, and export the data for usage in another software. While some of the integrated services are based on open source software, the WebLicht application (and most services) are closed source software.

Nopaque²⁵ is an open source web-based application tailored to work with historical documents. It includes a customizable pipeline from OCR (based on tesseract²⁶) over classic NLP tools (based on spaCy (Honnibal et al., 2020)) like tokenization, lemmatization, PoS tagging, NER, and indexing to allow syntactic analysis, KWIC search and complex search requests using metadata and extracted NLP information. Nopaque uses a standard client-server architecture with a central database and message queue. The web-based client provides a UI for the functionalities provided by the server.

iLCM²⁷ (Niekler et al., 2018) is an integrated research environment with a text mining focus to process large text collections with statistical analysis for research questions in social science. While it lacks modern NLP techniques, it features well-tested generic, high-performance statistical NLP methods (based on spaCy (Honnibal et al., 2020)) for large-scale data such as word frequencies, KWIC, co-occurrences, topic models together with customizable executable scripts (based on Jupyter Notebooks (Kluyver et al., 2016)) to address specific research requirements in the CSS and DH. iLCM allows manual annotations and training a classifier from the data. It is provided as a software as a service (SaaS) architecture with a web browser as the client and two main services interacting with each other: The Leipzig Corpus Miner (LCM) to preprocess and analyze large amounts of text, and the Open Research Computing

²³<https://mimno.github.io/Mallet/>

²⁴https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/Main_Page.html

²⁵<https://nopaque.uni-bielefeld.de>

²⁶<https://tesseract-ocr.github.io/>

²⁷<https://ilcm.informatik.uni-leipzig.de>

(ORC) environment to run scripts in executable documents (notebooks). Multiple users can simultaneously work on a shared set of documents in the platform.

Active learning (AL) is a widely discussed method to combine the strengths of humans and machines. While there are a multitude of tools, only few are easily and intuitively usable without programming knowledge and thus suitable for the application in DH or CSS contexts.

ActiveAnno (Wiechmann et al., 2021) is a simple open-source web tool with a responsive UI for mobile devices to annotate documents quickly, and train a simple classifier on short text snippets with an AL integration. It is designed for document-level annotations tasks, allows configuring the annotation setting, and includes features for adjudicating disagreements and analyzing annotation results. ActiveAnno uses a standard client-server architecture with a central database.

LabelSleuth²⁸ (Shnarch et al., 2022) is a sophisticated open-source software to train a binary sentence classifier from scratch without any expertise in programming or ML. It provides an intuitive UI for sentence annotations, features ML-based AL techniques to guide the user which sentences to label next, and allows configuring classifier models and AL strategies. However, it is unsuitable for multi-class or word-level annotations. LabelSleuth is build as a client-server architecture with a web-based client and server written in Python.

Summary of NLP methods in the tools Basic count-based NLP methods for DH and CSS are more numerous than neural methods and clearly the focus of most tools. This has multiple reasons: Software is often old, and the integrated automation components cannot easily be changed. Thus, the NLP methods in ready-to-use software is often outdated. Another reason is that these older methods are explainable and understandable by most DH and CSS researchers. This is not the case for the more modern neural methods, which also require significantly more computational resources. Most neural NLP methods found in available software are a direct replacement for standard tasks previously performed with classical, non-neural NLP such as NER. A niche are trainable (binary) classifiers that might use classical ML algorithms like support vector machines or even transformer-based neural networks (NNs).

3.2 NLP Methods in the Digital Humanities and Computational Social Science

A wide range of NLP methods have found their way to DH and CSS. As text analysis often is a central activity in DH workflows, NLP methods are often employed to work with more data to acquire knowledge and validate hypotheses. For a recent survey of NLP methods in social science refer to Hou and Huang (2025). This section provides an introduction to commonly used methods that are either highly related to methods developed in this work, referred to in later chapters of this work, or build the foundation for such methods. We will discuss the methods regarding their usage, applicability, and implementation as statistical, ML-, or SotA NN-based methods.

²⁸<https://www.label-sleuth.org/>

3.2.1 Data Exploration

Data exploration methods provide ways to deal with large quantities of textual data with little to no additional input or knowledge apart from providing the raw text of the corpus. This makes such methods generally appealing as a starting point in DH contexts. However, some of the methods can also be applied later in a refined way. Apart from topic modeling, all other methods in this section are purely count/frequency based operating directly on either the raw text or a lemmatized version.

Word counts are possibly the simplest form of a statistical text analysis to answer questions such as: “Which are the most common words?” Function words (determiners, prepositions, *etc.*) are placed at the stop unless they are excluded via so-called stop-word removal. The question of “How many words are in a corpus?” can be answered with either the number of word tokens or the number of word types that is how many different words are in a corpus. (Manning and Schütze, 1999, pp. 20–23) Count-based analysis features are implemented in many software applications, easy to understand, require no additional input except the raw text, and can be combined with other methods such as collocations, lemmatization, *n*-grams, or PoS tagging *etc.* Thus, count-based methods such as word counts they are still commonly relied upon in DH contexts.

Collocation analysis A collocation is a phrase of two or more words that often have a specific meaning or are otherwise idiomatic, *e.g. make up* or *crystal clear*. In any case, a collocation is a frequently-used, fixed expression. Collocations filtered by a particular word show frequent ways in which that word of interest is used. (Manning and Schütze, 1999, pp. 151–157) Collocations provide another potentially interesting view of a corpus, many DH tools support them, and require only the plain text as input. Consequently, collocation analysis is often applied in DH contexts.

Concordances & KWIC (Key Word in Context) KWIC is a method to display all occurrences of a specific word with a user-defined number of surrounding words on both sides; optionally sorted by the left or right context. It allows finding patterns in which the word is used in the given corpus. (Manning and Schütze, 1999, pp. 31–34) KWIC is heavily used in DH context for search and analysis. It is easy to understand, many DH tools provide a KWIC feature, making it a commonly-used method as it can be applied to a wide range of research questions.

Co-occurrences analysis Co-occurrences are words that occur in the same context and can be seen as collocations without being grammatically bound together. Co-occurring words simply tend to appear close to each other or in similar contexts, *e.g. doctor* and *hospital*. (Manning and Schütze, 1999, pp. 183–186) Co-occurrences can be easily computed by counting the number of any two words appearing in the same context such as a sentence by filling out a large matrix with rows and columns for each word type in the corpus. Similar to collocations, co-occurrences can provide an explorative overview of a corpus without any additional input, or can be filtered when only looking at the co-occurring partner words for a specific set of words. As many tools support finding co-occurrences, it is a frequently applied method in DH and CSS contexts.

***n*-grams** are successive sequences of words such as unigrams ($n = 1$), bigrams ($n = 2$) and trigrams ($n = 3$). While mostly used in NLP to create *n*-gram models predicting the next word given the $n - 1$ previous words, *n*-grams can also be used as a way to extract multi-word phrases for a manual analysis. (Manning and Schütze, 1999, pp. 191–196) Some tools provide the most frequent *n*-grams for a user selected value of *n* to identify frequent multi-word expressions *etc.* While possible, *n*-grams are rather seldomly used in DH or CSS contexts on their own as a method. Instead, proper keyword or key phrase extraction is applied as it is more user-friendly.

Topic modeling (Blei, 2012) are a family of methods to explore large text document collections by discovering (hidden) topics in the collection. They create statistical models from frequently co-occurring words in segments sampled randomly from the collection. The extracted topics are defined by sets of words such as {car, mechanic, oil, repair} or {children, play, kindergarten} allowing to explore large document collections quickly by only looking at the topics and seeing which documents belong to which topics. The arguably two most popular techniques for topic modeling are Latent Semantic Analysis (LSA) (Deerwester et al., 1990) and Latent Dirichlet Allocation (LDA) (Blei et al., 2003). Both are statistical models without any regard to the meaning of words. With the widespread application of NNs, neural topic modeling (NTM) has emerged with many neural models offering flexibility and scalability in efficiently handling large-scale datasets (Wu et al., 2024). Still, most software for DH and CSS research, relies on the older, statistical topic models. Topic modeling is widely used in DH and CSS as it is easy to apply via available tools, interpretable and greatly helps in textual data exploration.

3.2.2 Search

Given a large collection of documents, search methods allow finding the documents most relevant to a search query. In CSS and DH contexts, this could be *e.g.* names of entities or concepts. The methods introduced in this section are standard count-based/statistical methods except the semantic similarity search.

Lemmatization and stemming are two means to deal with morphology in language; inflected word forms sometimes create issues for the above-mentioned corpus-linguistic techniques, and pose significant problems for text-based search methods. Stemming strips of word affixes to produce a word stem that may not be a known word. Lemmatization is a more advanced approach that converts an inflected form back into its lemma or lexeme, the base form of a word. (Manning and Schütze, 1999, pp. 131–134) Lemmatization and stemming are commonly performed pre-processing steps where other algorithms operate on surface forms such as word counts, co-occurrences or search.

TF-IDF (Term Frequency-Inverse Document Frequency) (Manning and Schütze, 1999, pp. 539–544) is a family of weighting schemes for term frequencies and (inverse) document frequencies that measure the importance of a term (word) to a given document in a collection of documents. It can be used to extract words specifically relevant to a document or to create a term-based search engine to score a document's relevance to a search query. While TF-IDF might be rarely used for search directly in DH or CSS contexts, it provides means to automatically extract keywords from texts.

BM25 (Robertson and Zaragoza, 2009) is a ranking function used in search engines to measure the relevance of a document to a query. It is similar to TF-IDF, and mainly used in document retrieval. BM25 relies on the bag-of-words model where the order of words does not matter, and it does not consider the meaning of the query or matching documents. If CSS and DH tools provide a text-based search functionality, it is typically based on the BM25 algorithm such as implemented in the popular ElasticSearch software. As such, BM25 is (unknowingly) frequently used in DH and CSS contexts when full-text search is used.

Vector-based semantic search uses semantic text similarity (STS) based on the embeddings of documents, paragraphs or sentences to provide a semantic search. In the STS task, two or more texts are compared to determine how similar the texts are. While this can be solved explicitly with a bi-encoder being provided the two texts as inputs simultaneously and asked to output a single number between 0 and 1, such an approach does not scale at all when searching for similar texts in a large collection. For n texts, it would require $n^2/2$ expensive comparisons. The more efficient alternative is to embed all texts once with a suitable embedding model to obtain sentence, paragraph or document embeddings that capture semantic information. Similar to contextualized word embeddings (CWEs) it is then possible to compute *e.g.* cosine similarity between the embedding vectors to calculate how close they are in vector space; or how similar they are semantically. For an efficient real-time search, all embeddings are indexed in an approximate nearest neighbors (ANN) index that enables fast retrieval of the nearest vectors given a search vector. Depending on the search, either a pre-computed query is used (*e.g.* for finding the most similar documents) or a new embedding is created on the fly by the same embedding model (for free form text search). The MTEB: Massive Text Embedding Benchmark (Muennighoff et al., 2023) and its successor the MMTEB: Massive Multilingual Text Embedding Benchmark (Enevoldsen et al., 2025) provide frameworks to evaluate embedding models across many languages, domains and tasks including STS. On MMTEB, current SotA embedding models like Qwen3-Embedding-8B (Zhang et al., 2025) reach a Spearman correlation based on cosine similarity of 0.8 — a strong correlation. As such, STS cannot be solved perfectly with embeddings yet, but good enough to perform well in a wide range of applications. While search is generally an important feature, none of the above discussed tools offer a vector-based search that could provide a large quality improvement over the dominant implementation based on BM25.

3.2.3 Extracting Structured Information

Techniques to extract structured information from texts such as keywords, key phrases and named entities are often employed in DH and CSS contexts and found in many tools. Automatically extracted key phrases and frequent named entities provide valuable information to understand a corpus by listing the most frequent actors mentioned in it (*e.g.* people, organizations, locations) as well as phrases specific to the domain (concepts) (Mcgillivray et al., 2020).

Part of speech (PoS) tagging describes the process of classifying each token into its PoS category such as noun or verb (Manning and Schütze, 1999, pp. 341–345). It is an essential pre-processing step in a classic NLP pipeline. Lemmatization algorithms often rely on PoS information of a word. PoS tags can be used to augment and refine above corpus linguistics

techniques, *e.g.* filtering word counts or keywords by their PoS tags. There are different PoS tag sets that can be language specific and fine-grained with sub-categories or coarse and multilingual. Universal PoS tags²⁹ defines a commonly used tag set split in three groups: *Open class words* are ADJ adjective, ADV adverb, INTJ interjection, NOUN noun, PROPON proper noun and VERB verb. *Closed class words* with a fixed list of words per class are ADP adposition, AUX auxiliary, CCONJ coordinating conjunction, DET determiner, NUM numeral, PART particle, PRON pronoun and SCONJ subordinating conjunction. The *Other* group contains PUNCT punctuation, SYM symbol and X other. SotA methods for PoS tagging are based on pre-trained language models (PLMs) and reach quality levels very close to 100% on clean texts. Thus, full automation is feasible in many cases and included in some commonly used software packages such as *spaCy* (Honnibal et al., 2020). However, PoS tagging on its own has only little value to most researchers. Instead, it is applied together with other methods as a filtering option *etc.*

Named-entity recognition (NER) is a technique to automatically detect variable-length spans of tokens referring to a named entity in a text (see *e.g.* Jehangir et al. (2023) for a more complete survey). It is commonly implemented as a sequence tagging task using the BIO notation which distinguishes the beginning (B) and inside (I) of entities while O (outside/other) is applied for all non-entity tokens. The CoNLL 2003 shared task (Tjong Kim Sang and De Meulder, 2003) started a competition of ML systems on a coarse tag set with four different entity types: PER person, LOC location, ORG organization and MISC miscellaneous. SotA Transformer-based models achieve high F1 scores above 94 and multiple works corrected the test set labels to improve evaluation (Rueda et al., 2024). The OntoNotes dataset (Weischedel et al., 2013) uses a finer-grained tag set with 18 classes. The performance of SotA models on the commonly used data split by Pradhan et al. (2013) reached over 90 F1 using the fine-grained classes. Current SotA approaches based on PLMs reach human-level quality depending on the domain of a text, making reliable automation feasible in many cases. NER is included in many tools, easy to understand and verify, and of great help for many DH and CSS researchers. Consequently, it is commonly used in CSS and DH settings, both for information extraction and network creation on a corpus level as well as automatic highlighting during close reading.

Keywords Keywords and key phrases can *e.g.* be extracted by applying TF-IDF on *n*-grams, possibly using lemmatization as an additional pre-processing step. They provide an extractive summary of a document *e.g.* for visualization as a word cloud or simple list. They further allow to find the words most relevant for an entire document collection when comparing with a background corpus. Keywords can be reliably extracted by statistical methods, but there are also neural models to extract or generate keywords using *e.g.* generative LLMs (Grootendorst, 2020). Keyword extraction is widely available in tools and commonly employed in CSS and DH contexts to reduce (long) documents to an amount of text quickly manageable *e.g.* by displaying as a word cloud or to use keywords for filtering for search or analysis methods.

Network graphs Network graphs provide another overview of a corpus. Nodes represent relevant phrases (*e.g.* actors and concepts), and the edges represent either a relation between the terms (*e.g.* support, oppose), or a notion of proximity between them (based *e.g.* on the overlap between their contexts). The automatic creation of network graphs is rather specific and not included in generic DH or CSS tools. Yet, DH and CSS researchers often resort to a

²⁹<https://universaldependencies.org/u/pos/>

network analysis and thus manually create the network and quantify the relations. (Mcgillivray et al., 2020) There are a few tools that provide automatic network graph as their main feature such as network of the day and new/s/leak (see the list at the beginning of Section 3.1.2).

Summary of NLP methods in DH contexts In this section, we discussed available NLP methods for data exploration, search and structured information extraction in DH and CSS contexts. The majority of the methods are simple, word count-based approaches due to availability in tooling, ease of application and understandability. Some frequently applied methods such as topic modeling, NER and keyword extraction are also available as neural models. However, search functionality is still limited to statistical approaches while vector-based semantic search is available in theory. While methods such as network graphs are highly desired in DH and CSS, they are rarely implemented as automatic approaches in available software.

Part II

Broadly Applicable NLP for DH & CSS Workflows

Abstract We present novel natural language processing (NLP) methods that are broadly applicable in many Digital Humanities (DH) and Computational Social Science (CSS) contexts; to assist researchers with large-scale text collections to answer their research questions. We describe methods to improve semantic similarity search for the DH, detect and attribute quotations, perform coreference resolution in German, and semi-automatically scale manual annotations.

In Chapter 4, we present methods for DH researchers to apply semantic similarity search in their workflows: A configurable similarity search providing insights into which aspects of documents are similar, and a method to search for sub-sentence text spans using semantic search as a modern alternative to KWIC (Key Word in Context). In Chapter 5, we work on quotation detection and attribution by creating a new manually-annotated dataset, and developing three approaches: A rule-based system, an adaptation of an existing machine learning (ML)-based system, and our novel neural ML model with highly superior quotation detection and attribution. In Chapter 6, we present two neural coreference resolution models for German that improve tremendously on previous systems while being computationally efficient even for long documents. Finally, in Chapter 7, we show a way to semi-automatically scale manual annotations to large corpora with our envisioned application and interactive method based on contextualized word embeddings and k -nearest neighbors.

4 Semantic Similarity Search for the Digital Humanities

Abstract In this chapter, we present two methods to help Digital Humanities (DH) practitioners to apply modern semantic similarity search in their workflows. After introducing the issues with off-the-shelf semantic search, we first describe a configurable similarity search providing insights into which aspects of the documents are similar. Second, we present a method to search for sub-sentence text spans using semantic search as a modern alternative to KWIC (Key Word in Context).

This chapter is (partly) based on the following publication(s):

Steffen Remus, Gregor Wiedemann, Saba Anwar, **Fynn Petersen-Frey**, Seid Muhie Yimam, and Chris Biemann (2022). “More Like This: Semantic Retrieval with Linguistic Information”. In: *Proceedings of the 18th Conference on Natural Language Processing (KONVENS 2022)*. Ed. by Robin Schaefer, Xiaoyu Bai, Manfred Stede, and Torsten Zesch. Potsdam, Germany: KONVENS 2022 Organizers, pp. 156–166. URL: <https://aclanthology.org/2022.konvens-1.19/>.
Hans Ole Hatzel, **Fynn Petersen-Frey**, Tim Fischer, and Chris Biemann (2023). “Dimensions of Similarity: Towards Interpretable Dimension-Based Text Similarity”. In: *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*. Ed. by Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu. Vol. 372. Frontiers in Artificial Intelligence and Applications. Kraków, Poland: IOS Press, pp. 1004–1011. DOI: 10.3233/FAIA230372.

4.1	Introduction	46
4.1.1	What is Similarity in Embeddings	46
4.1.2	Providing Explainability	47
4.1.3	Goal and Outline	47
4.2	Related Work	47
4.2.1	General Document or Sentence Similarity	47
4.2.2	Aspect-based Similarity	48
4.2.3	Structured Similarity	49
4.3	Configurable Similarity Search	49
4.3.1	Experiments using the SemEval 2022 Task 8 Dataset	51
4.3.2	Application-emulating Experiments	55
4.4	Structured Similarity Search	58
4.4.1	Methodology	58
4.4.2	Linguistic Structures	60
4.4.3	Experiments	61
4.4.4	Results	62
4.5	Conclusion	64

4.1 Introduction

Traditionally, information retrieval methods for search primarily relied on string matching on keywords or word count-based measures. More recently, semantic search based on embedding similarity measures in a vector space has seen an increase in popularity. Refer to Section 3.2.2 for more information on existing search methods. Current semantic similarity models (e.g. Reimers and Gurevych (2019)) are based on the successful Transformer (Vaswani et al., 2017) model using Siamese training setups (see e.g. Chicco (2021) for an overview) that generate embeddings especially suited for the task of similarity search.

Yet, search in DH tooling is typically still very basic from a natural language processing (NLP) perspective: Fixed string matches and keyword-based approaches dominate. Sometimes, slightly more advanced methods including lemmatization and TF-IDF (Term Frequency-Inverse Document Frequency) methods or BM25 are employed for full document searches. KWIC is frequently used to find the exact occurrences of a specific search term and provide matches with the surrounding context. However, we repeatedly found that newer, semantic search is rarely used in interdisciplinary DH contexts. When suggesting semantic similarity search in these contexts, we repeatedly encountered questions of interpretability and granularity. In essence, the DH researchers are hesitant to apply semantic similarity search in their work due to two main reasons:

1. It is not explainable how such a search method actually works,
2. or there is no semantic search equivalent available (in case of KWIC).

4.1.1 What is Similarity in Embeddings

Neural language models (NLMs) producing contextualized word embeddings (CWEs) such as ELMo (Embeddings from Language Models; Peters et al., 2018), Flair (Akbik et al., 2018), or BERT (Bidirectional Encoder Representations from Transformers) (Peters et al., 2018), or one of the many derived successors have greatly improved the performance on almost any NLP task. One of the major reasons for this effect is that these recent NLMs are able to produce compositional representations of a word in a vector space based on the individual context in which it appears. Thus, the embeddings of two surface forms used in a similar context will be close in vector space. The surface forms of a word with two (or more) meanings will be represented by embeddings further apart in vector space. This property naturally handles paraphrases based on synonyms. What CWEs enable for a word, sentence or document embeddings based on similar techniques allow creating vector representations of sentences (and to some degree entire documents) that sufficiently represent the compositional meaning of all words in it.

Sentence BERT (Reimers and Gurevych, 2019) and related approaches focus on textual similarity in the sense of paraphrases. The models are trained on large datasets containing paraphrases of different degrees. With this training, they enable search in the vector space by looking for the smallest distance between embeddings. Despite the sharp increase in search result quality, practitioners want interpretability and explainability of similarity according to our experience when working in interdisciplinary projects with DH or Computational Social Science (CSS) researchers.

4.1.2 Providing Explainability

Unlike traditional, word count-based methods used in DH or CSS, understanding why two documents (or a query and a document) are considered similar based on their embeddings in the vector space is non-trivial. While it intuitively follows from the Siamese training objective (Chicco, 2021) of current semantic similarity models (Reimers and Gurevych, 2019) that paraphrases of the same sentence are deemed similar, it is not always clear how this property applies to any given pair of texts. Making semantic search based on embeddings more explainable has to be tackled on two fronts:

- Explaining how semantic search works to the users of a system that provides such a functionality,
- Listing what (e.g. which dimension or aspect) is considered for two texts to be similar and how similar the different aspects are (providing help to understand what a similarity of e.g. 0.7 is).

While the first point is out of scope for this work, we work towards the second point by performing an evaluation what current model focus on and creating a method to let the user configure what the similarity model should focus on.

4.1.3 Goal and Outline

To tackle the blockers due to which DH researchers do not apply semantic similarity search, the goal is to, first, create a semantic search that is less black-box than currently available off-the-shelf systems, make searching in the embedding space more explainable resp. configurable and, second, work towards a semantic search for precise, short matches.

In the following sections of this chapter, we first review related work on general document similarity, aspect-based similarity and structured similarity. Then, the methodology to approach the first goal is described before performing experiments on an existing dataset to create search models for a more explainable and configurable embedding space similarity. Subsequently, we apply these models and test the configurable similarity search on independent dataset and tasks. In the next section, we tackle the second goal and present an approach for a structured similarity search to work towards a semantic search for short matches. Finally, we conclude our findings and reflect on the limitations of both the methods and experiments performed.

4.2 Related Work

In the following sections, we provide a review of the three different topics related to our envisioned semantic similarity search for DH. First, we review general document or sentence similarity as the foundation of a typical semantic search. Next, we look into existing approaches for an aspect-based search that is critical for a more controlled semantic search. Finally, we review work on structured similarity as a step towards semantic search on sub-sentence text spans.

4.2.1 General Document or Sentence Similarity

Two currently popular architectures for text similarity comparisons are cross-encoders and bi-encoders. In both approaches, a language model is used. In a cross-encoder setup, one

model receives both documents as input and produces a similarity score, whereas, in a bi-encoder, only one document is encoded at a time with the similarity being computed by comparing the output representations of various documents using an applicable similarity scoring function such as Euclidean distance or cosine similarity. As is already evident by this description, a cross-encoder can only compare two documents at a time, whereas the output representation comparison of the bi-encoder easily scales to more documents making it the suitable architecture for corpus-wide similarity search. While bi-encoders have been shown to perform slightly worse than cross-encoders (Reimers and Gurevych, 2019), the quadratic time requirement of cross-encoders for corpus-level comparisons makes bi-encoders the only suitable choice for efficient similarity searches on large corpora, although in the SemEval task itself participants came to different results when comparing the performance of the two architectures (Chen et al., 2022).

SentenceBERT is a Siamese network-based (Reimers and Gurevych, 2019) approach for semantic sentence embeddings. It is an extension to pre-trained transformer architectures such as BERT or RoBERTa, which is specifically targeted for sentence similarity search, *i.e.* finding similar sentences by using cosine similarity. SentenceBERT outperforms most other embedding strategies for multiple sentence similarity tasks. However, it requires labeled data in form of similar/dissimilar sentences. In the recent past, various adaptations of the approach, such as SentenceBERT’s language-agnostic counterpart LaBSE (Feng et al., 2022) have become quite popular in semantic search applications. The produced sentence embeddings allow a generic, blackbox-like semantic search across different languages using cosine similarity.

4.2.2 Aspect-based Similarity

The SemEval 2022 Task 8 shared task (Chen et al., 2022) introduced a dataset of annotated news links, with the corresponding articles being scrapable from the Web. It consists of about 10,000 news article pairs of 18 language combinations. As the first of its kind, the dataset introduces seven dimensions of similarity annotated for each pair of articles: *geography* (GEO), *entities* (ENT), *time* (TIME), *narrative* (NAR), *style* (STYLE), *tone* (TONE), and *overall* (OVERALL). In each pair, the scalar similarity along each dimension was rated as a whole number on a four-point scale (1–4) by multiple human annotators, with the average of multiple annotators being used as the gold target. As randomly sampled article pairs are very likely to be entirely unrelated, various sampling techniques, such as the overlap in named entities, were employed to provide annotators with article pairs that have some degree of similarity. The dataset has two splits for evaluation: “English only” and “All data”. Task participants were only evaluated on their performance in predicting the OVERALL dimension. The limited evaluation may, in part, have been motivated by the strong correlation between some classes; the scalar value describing the NAR class, for example (which focuses on narrative schema similarity), shares a Pearson correlation of $\rho=0.88$ with the OVERALL class. For other class combinations, however, we see correlations as low as $\rho=0.22$ (TIME and STYLE). Most participants did not utilize the extra dimensions, while few others (Xu et al., 2022; Goel and Bommidi, 2022; Hajjar et al., 2022) experimented with multi-label loss based on the additional dimensions. Only Goel and Bommidi (2022) reported scores for other dimensions (in the form of mean squared error loss), but their final correlation score was not competitive. Both bi-encoders and cross-encoder

architectures were submitted as part of the task, with no clear winning architecture emerging in the evaluation of the system performances (Chen et al., 2022).

Ostendorff et al. (2020) take a different approach to aspect-based similarity in scientific papers by comparing sections with the same titles using sentence similarity models, resulting in aspects of similarity such as *Results* or *Related Work*. Prior work exists in configurable search approaches (e.g. Köppelmann et al., 2012), but we do not know of further work in the context of neural semantic similarity search.

4.2.3 Structured Similarity

The LISA (Linguistically-Informed Self-Attention) approach by Strubell et al. (2018) showed the benefit of injecting syntactic information into a neural network (NN) using self-attention for multi-task learning (MTL). LISA was applied for dependency parsing, part of speech (PoS) tagging, predicate detection, and semantic role labeling (SRL), where the results for all tasks showed significant improvements over the previous state-of-the-art, particularly when using ELMo embeddings (Peters et al., 2018).

Wiedemann et al. (2019) showed that contextual embeddings, particularly BERT (Devlin et al., 2019) inherit a certain degree of sense representation, *i.e.* polysemous words appear in different areas of the embedding space depending on their context. Wang et al. (2019) implement Elman (1990)’s theory, which states that NLMs are sensitive to word order regularities in simple sentences, by specifically exploiting the inner-sentence structure (word-level ordering) and inter-sentence structure (sentence-level ordering) as training objectives. They argue that their StructBERT model successfully captures the structure of sentences during pre-training. This is achieved by pre-shuffling trigrams that are not used as masked tokens within sentences and using three different sentence prediction objectives: next sentence prediction, previous sentence prediction and random sentence prediction.

Htut et al. (2019) and Clark et al. (2019) analyze to which extent attention heads in BERT can track linguistic dependencies. Both works conclude that some attention heads specialize in syntactic structure. Wu et al. (2020b) measure the impact one word has on another in a sentence by using a so-called perturbed masking technique. They can derive a syntax tree from a word-word matrix. Soares et al. (2019) used a so-called masking technique to specifically force the model to learn entity locations in a sentence. By doing so, specific representations for particular relations within text can be learned.

4.3 Configurable Similarity Search

Overview We approach the first goal of a configurable similarity search system by modeling various dimensions of similarity, giving the user insight into which aspects of the documents are similar. Toward our goal, we develop various models that focus on different aspects of the texts trained with the dataset provided as part of the SemEval 2022 Task 8 (Chen et al., 2022), which contains news articles that were annotated with regard to their similarity along seven dimensions. We exploit these annotations of seven dimensions of similarity to train models that should be able to represent documents according to these different aspects (see Section 4.3.1, page 51). Additionally, we further analyze the dimensions and test the broader application to retrieval-focused downstream tasks (see Section 4.3.2, page 55).

Similarity search

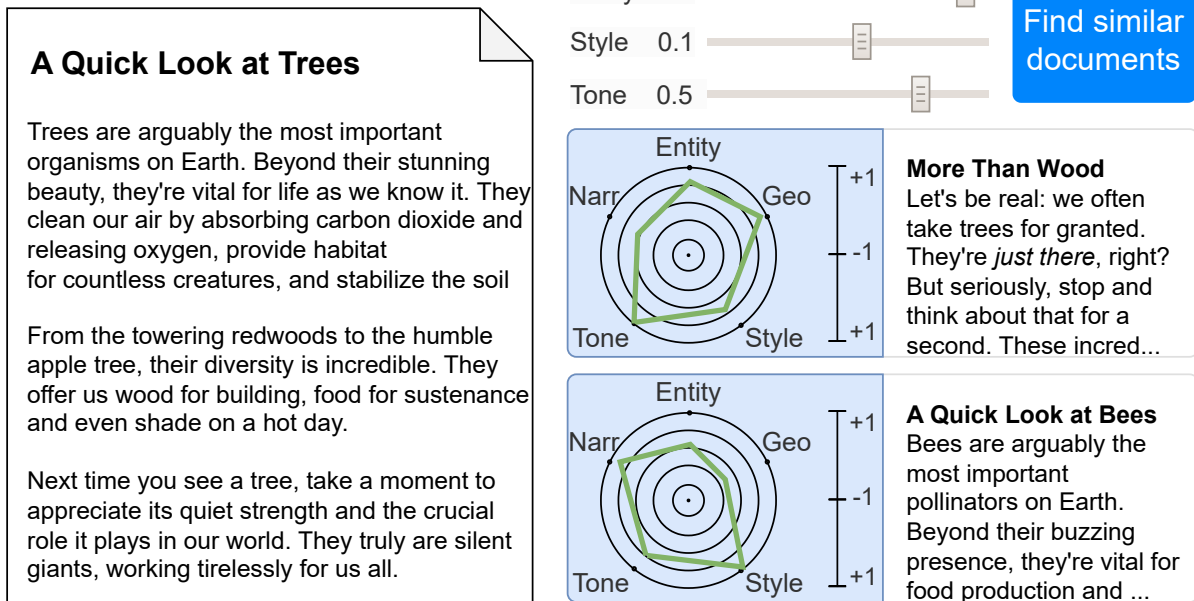


Figure 4.1: Search mockup. Left: current document. Right: configurable search and results that visualize its similarity across the various dimensions. Adapted from Figure 1 in Hatzel et al. (2023). Example texts are generated with the large language model (LLM) gemma-3-27b-it (Team et al., 2025).

Similarity analysis

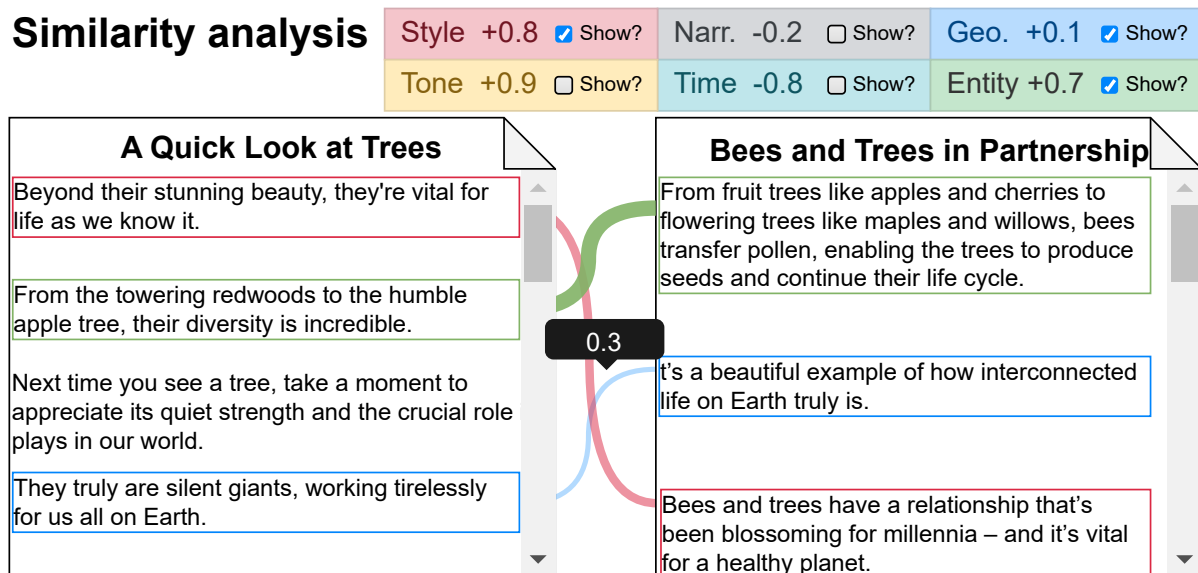


Figure 4.2: Document similarity analysis mockup. The similarity of two documents is broken down to sentence-wise dimension-based similarity. Adapted from Figure 2 in Hatzel et al. (2023). Example texts are generated with the LLM gemma-3-27b-it (Team et al., 2025).

Envisioned application We envision an application that allows for a configurable similarity search (see Figure 4.1) to focus on different aspects — for example, to retrieve documents that cover the same topic but from a different perspective as manifested in the style or tone — and brings interpretability to the results of similarity search (see Figure 4.2). As we envision a retrieval system that scales to large document collections, we consider only bi-encoders in our experiments. Approximate nearest neighbors (ANN) indices of document embeddings are typically utilized to develop fast and efficient document retrieval systems. While models with bi-encoder architecture can compute document embeddings, the cross-encoder architecture is not suitable for building such indices. More concretely, an application of our proposed approach could be found in a configurable similarity search where users, for example, retrieve documents on the same topic but covering it from a different perspective as manifested in the style or tone of the document.

Outline We train an individual model for each dimension of similarity as annotated in the SemEval dataset (see Section 4.3.1.1, page 51). This experiment validates whether sentence similarity models are able to capture these dimensions and the resulting models could possibly be used in our envisioned application. However, we understand that having seven different models requiring seven inference steps per document to obtain the different document representations is rather inefficient. This is the reason why, in the next step, we evaluate the use of MTL to develop a single model that produces embeddings for each dimension in a single forward pass. We achieve this by using a dimension-specific head that transforms the final embedding. Afterwards, we compare human judgements and machine judgements in distinguishing the different dimensions (see Section 4.3.1.2, page 53). Finally, we move on to application-emulating experiments (see Section 4.3.2, page 55) to evaluate the use of our trained models on out-of-domain data. These experiments further serve as an exploration of the potential for configurable similarity search.

4.3.1 Experiments using the SemEval 2022 Task 8 Dataset

We evaluate the performance of pre-trained models on the SemEval 2022 Task 8 data and finetune existing models along all seven dimensions of similarity annotated in the dataset: *geography* (GEO), *entities* (ENT), *time* (TIME), *narrative* (NAR), *style* (STYLE), *tone* (TONE), and *overall* (OVERALL). The finetuned models will be evaluated in detail for our configurable similarity search.

In preliminary experiments, we tested multiple pre-trained foundation models to use for our main experiments. Following the results on the shared task (Chen et al., 2022), we selected LaBSE as it performs best on multilingual data.

4.3.1.1 Finetuning

Finetuning was performed in a Siamese-network setup, training on a cosine similarity loss using mean squared error to compare the cosine similarity of the Siamese network outputs with the training label, incentivizing (cosine-)similar embeddings for documents that were annotated as similar. We use the SentenceTransformer (Reimers and Gurevych, 2019) library to perform the finetuning and set the training parameters to 3 epochs, 100 warm-up steps and batch size of 32. The learning rate and optimizer were kept as default values of 2×10^{-5} and AdamW. In the labeled data, 1 represents high similarity along a specific dimension, whereas

LaBSE Variant	GEO	ENT	TIME	NAR	STYLE	TONE	OVERALL
pre-trained	.44	.64	.44	.65	.37	.44	.65
multimodel	.62	.77	.52	.78	.56	.53	.78
MTL	.53	.74	.51	.78	.48	.51	.76

Table 4.1: Pearson correlation with human judgements of our two approaches on the multilingual evaluation split. The multimodel finetuned LaBSE models outperform the MTL model and the pre-trained LaBSE on every dimension.

4 represents extreme dissimilarity. We found that transforming the label scale had a large impact on performance and thus followed Di Giovanni et al. (2022) in this regard. In fact, introducing this label-scaling scheme was the single change, outside pre-trained model choice, that we found to have by far the largest positive impact on performance. Using Equation (4.1), we transform the labels from the $[1, 4]$ range to $[0.0, 1.0]$ instead of $[-1.0, 1.0]$, as one would assume for cosine similarity.

$$f(x) = \frac{4 - x}{3} \quad (4.1)$$

We approach our goal, to obtain models that assess the similarity of documents with respect to seven dimensions of similarity, from two directions. Our first multimodel approach is to finetune one model per dimension of similarity. There was no need to change the model architecture of LaBSE for this approach. It results in 7 different models, each specialized for a certain dimension.

However, for our envisioned application of large-scale, configurable similarity search, this setup could be more efficient. Using individual models, each having its own set of weights, would require multiple inference runs to obtain all dimension-specific document embeddings.

Accordingly, we attempt in our second MTL approach to unify all dimensions into a single MTL model. Here, we changed the model architecture of LaBSE by adding multiple heads (one for each dimension of size 512). Further, we altered the training objective to optimize for all dimensions at once, while including an example of each dimension in each batch and keeping all other training parameters the same. This setup has the advantages of sharing the vast majority of parameters as well as requiring only a single inference run to obtain all dimension specific document embeddings.

We evaluate these two approaches on the multilingual evaluation split and compare it to the unfinetuned LaBSE (pre-trained). The reported scores denote the Pearson correlation with human judgements. Results are shown in Table 4.1.

We observe a clear improvement in LaBSE’s performance when finetuning on each of the individual dimensions. For the GEO, ENT, NAR, OVERALL and STYLE dimensions there is a vast improvement over the pre-trained model. TIME and TONE, however, are still worthy of improvement.

For the MTL model, we see that for the dimensions OVERALL and NAR the performance is essentially equivalent, but it lacks far behind on other dimensions. We attribute this to the

fact that the MTL model has trouble separating the different dimensions as the OVERALL and NAR dimensions are strongly correlated (and to a lesser extent also ENT) — a hypothesis that is supported by our results in Section 4.3.1.2.

The best participant in the shared task (Xu et al., 2022) achieved a Pearson correlation of 0.818 with OVERALL on the multilingual evaluation split using a cross-encoder architecture. However, for our use case of similarity search and retrieval, only the bi-encoder architecture is relevant. In this category, the best system by Singh et al. (2022) also used LaBSE in combination with data augmentation techniques as well as metadata-based features and achieved a Pearson correlation of 0.801 with OVERALL, which is only slightly above ours of 0.78 while using a considerably more complex system than our finetuned sentence embedding model. As a result, we use our own model for all further experiments in which we typically do not have access to all the same metadata (such as a release date) as in the SemEval setup.

4.3.1.2 Comparing Human Judgements and Machine Judgements

In this section, we analyze whether machines or humans can better distinguish between the different dimensions. Further, we try to identify which dimensions machines focus on compared to human judgements. To do so, we look at the correlations between all pairs of dimensions. For human judgements, we use the labels from the dataset to produce the pairwise Pearson correlations as shown in Figure 4 of Chen et al. (2022). For machine judgements, we calculate the pairwise Pearson correlations between two dimensions of two models finetuned on these dimensions. We performed this both for the individually finetuned models shown in Figure 4.4 and the MTL model shown in Figure 4.5. In case of the MTL model, the dimension-specific embeddings are used in the pairwise comparisons. Finally, we subtract the human judgement correlations from the machine correlations resulting in Figure 4.6 to identify how machines results differ from human judgements.

It is clearly visible that the machine judgements have a higher inter-dimensions correlation than human judgements, *i.e.* all values are positive. Further, it is most obvious that the MTL model has an even higher inter-dimensions correlation than the individual models. When looking at the individual models, it can be seen that they are especially worse at distinguishing STYLE and TONE than humans are. The visual outliers with the smallest correlation differences (OVERALL–NAR, ENT–NAR, ENT–OVERALL), are cases where the human judgements already strongly correlate. The same (but slightly larger) effect also can be observed for the models. From these experiments, we conclude that the finetuned models are worse at distinguishing the different dimensions than humans are. While the individual models are only slightly worse than the human judgements, the MTL model is almost unusable in this regard. More specifically, the inter-dimension correlation for the MTL model is extremely high (> 0.9) for all pairs. We verified this extensively to ensure it is not a bug in our code. We attribute this to the fact that the model cannot really change the majority of parameters to deal with specific dimensions. It is thus forced into a very similar behavior for all dimensions. Potential future remedies may be adjusted training curricula, as we currently have one example of each dimension in each batch. Another potential avenue of improvement is the use of adapters (Houlsby et al., 2019) which would allow for sharing most parameters and only using a small set of dimension specific ones. Further improvements may be achievable by using adapter fusion (Pfeiffer et al., 2021) which in principle allows for sharing model capabilities across dimensions but would still require an inference pass for each dimension.

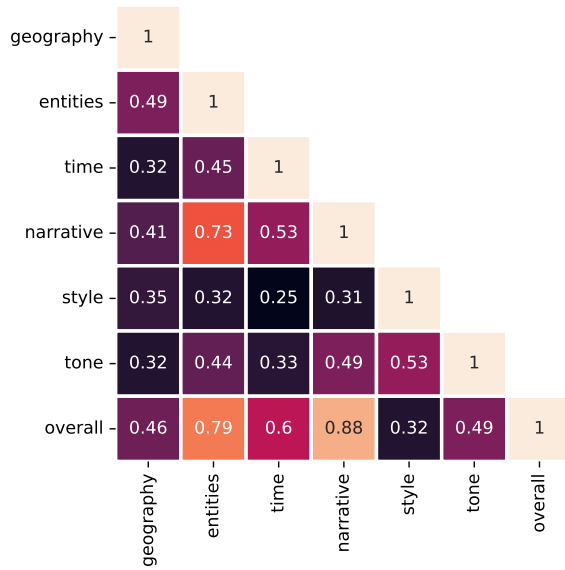


Figure 4.3: Correlation between the dimensions of human judgements from the dataset.

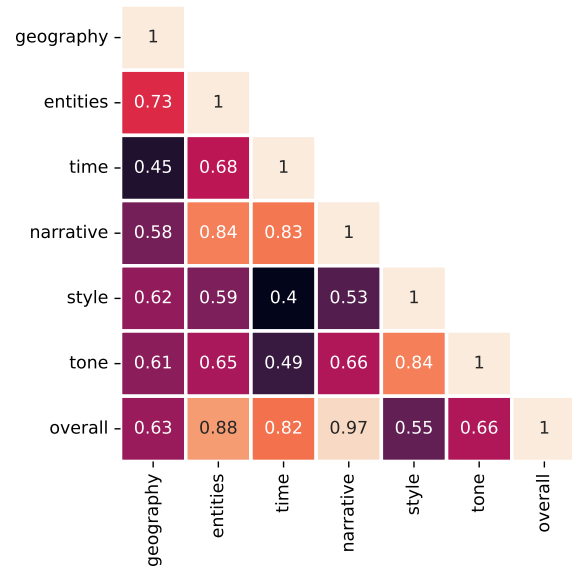


Figure 4.4: Correlation between the dimensions of the individually finetuned models, taken from Hatzel et al. (2023).

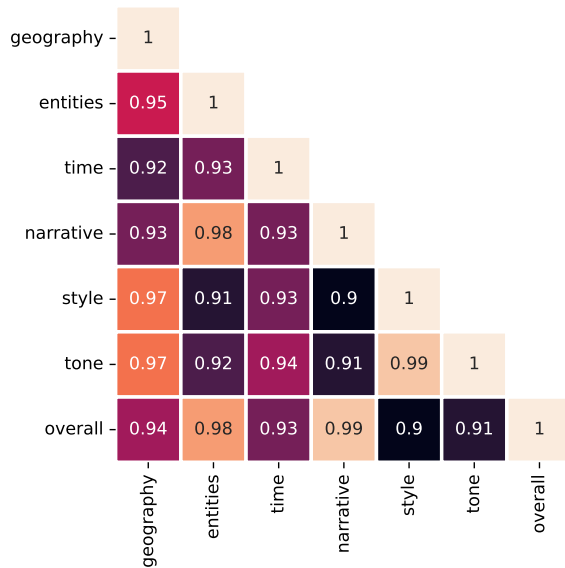


Figure 4.5: Correlation between the dimensions of the MTL model far surpassing the individual models, taken from Hatzel et al. (2023).

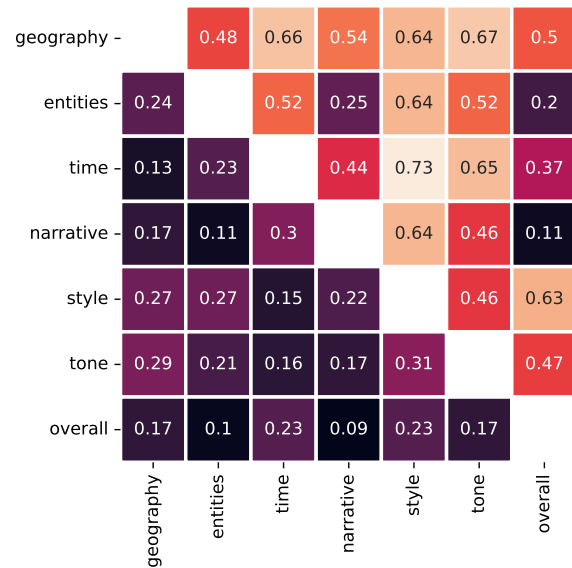


Figure 4.6: Correlation differences to human correlations: MTL model is on the top right, individual models are on the bottom left, taken from Hatzel et al. (2023).

4.3.2 Application-emulating Experiments

In the typical application of similarity search, paraphrase models are implicitly aligned with the users' intentions. In research applications, however, we observed that practitioners often desire a precise understanding of which aspects of texts are actually similar. Our approach is to model these aspects using the dimensions as annotated in the SemEval 2022 Task 8 dataset.

We envision an application used by domain experts to retrieve documents fulfilling certain similarity criteria. Such criteria could be configured by assigning weights to each dimension. The user could be presented with sliders controlling the similarity in each dimension ranging from -1, to +1 as depicted in Figure 4.1. As part of an exploratory research process, such an application would give instant feedback to changed weightings by displaying only those documents that score highest with respect to a current anchor document. For example, the user would be able to retrieve news documents covering the same incident from different perspectives by looking for high similarity in the entity and time dimensions but low similarity in the style and tone dimensions.

One example of such an application could be finding documents covering the same topic but conveying opposing points of view. In this specific example, we model the same incident by means of positive weights for the overall and entity dimensions and the different perspectives by a negative weight for the tone dimensions.

This task can be formulated as a weighted sum of similarities along dimensions of interest. More formally: given a pair of documents p and a weight vector w ($w \in \mathbb{R}^D$) that assigns a weight to each dimension $d \in D$, the weighted similarity $t(p, w)$ over all dimensions D is computed as follows:

$$t(p, w) = \sum_d w_d \cdot s_d(p) \quad (4.2)$$

where the similarity of a pair of documents in a certain dimension $s_d(p)$ can be estimated by our finetuned models described in the previous section.

In the following, we conduct experiments on two datasets, applying the suggested scheme from above. For all following experiments, we use the best versions of our LaBSE models that were finetuned on a single dimension. For these experiments, we assume correlations between specific dimensions in our finetuned model and labels created in other datasets. These assumed correlations are based on intuitive judgements and were not programmatically optimized in an effort to show the potential application of configurable searches on arbitrary data. The goal of all following experiments is to understand how the dimensions relate to annotations in existing datasets, *e.g.* in our first experiment (Section 4.3.2.1), we analyze if the TONE dimension is sufficiently correlated with ratings of product reviews. In our second experiment (Section 4.3.2.2), we analyze if the STYLE dimension sufficiently correlates with the document category. We are essentially modeling downstream applications for unseen datasets, simulating various use cases. Showing positive results in these experiments indicates that a configurable similarity search is feasible.

4.3.2.1 Product Reviews Experiments

For this first downstream experiment, we use the review dataset by Fang and Zhan (2015). This dataset is a set of over 5 million product reviews collected from Amazon in 2014. Each

Finetuned	Model Name	mAP
✗	random	0.421
✗	LaBSE	0.451
✓	LaBSE-tone	0.458
✓	tone - overall	0.447
✓	2 · tone - overall	0.464

Table 4.2: Mean average precision (mAP) scores for retrieving a review of different products with the same star rating.

Finetuned	Model Name	mAP
✗	random	0.006
✗	LaBSE	0.036
✓	LaBSE-overall	0.089
✓	overall + entities	0.088
✓	overall + entities - tone	0.090

Table 4.3: mean average precision (mAP) scores for retrieving a review for the same product with a different rating.

product review has a rating that is based on a star-scaled system, where the highest rating has five stars, and the lowest rating has 1 star, generally meaning ‘I love it.’ and ‘I hate it’, respectively. We seek reviews on the same product but with different ratings and vice versa. While originally conceived for sentiment analysis task, we repurpose the dataset to simulate the following retrieval tasks:

1. Given a review with a certain star rating, we want to find reviews with the same rating for other products.
2. Given a review of a certain product, we want to find reviews of the same product, but with different ratings.

As the search space, we select the first 100 products that have 100 or more reviews and their first 100 reviews totaling 10,000 documents. We model the same product using the ENT and OVERALL dimensions, and we model the same star rating using the TONE dimension. In other words, when applying our finetuned models in this experiment, we assume increased similarity in the ENT and OVERALL dimension for reviews of the same product. In contrast, we expect increased similarity in the TONE dimension for reviews with the same star rating.

Table 4.2 shows the results of Task 1. Indeed, our model finetuned on the TONE dimension (finetuned-LaBSE-tone) outperforms the pre-trained LaBSE and random baseline slightly. Additionally, we configured the similarity search to lessen the impact of the OVERALL dimension by subtracting it, denoted as ‘tone - overall’, further improving the performance only when the TONE dimension is given more weight, denoted ‘2 · tone - overall’.

Table 4.3 shows the results of Task 2 and has throughout significantly lower scores than Table 4.2 as the number of products is much larger than the number of ratings. This is also

Model Name	mAP	r-precision	p@5	p@10	p@100
finetuned-LaBSE-tone	0.458	0.459	0.356	0.37	0.441
finetuned-LaBSE-entities	0.437	0.437	0.308	0.317	0.397
finetuned-LaBSE-overall	0.436	0.437	0.301	0.311	0.39
sentence-transformers/LaBSE	0.451	0.447	0.402	0.415	0.463
tone - overall	0.447	0.451	0.366	0.374	0.421
tone - overall - entities	0.422	0.423	0.357	0.361	0.394
tone - entities	0.443	0.448	0.401	0.401	0.42
2 · tone - overall	0.464	0.467	0.464	0.468	0.477
random	0.421	0.421	0.419	0.419	0.418

Table 4.4: Precision scores for retrieving a different product with the same star rating.

Model Name	mAP	r-precision	p@5	p@10	p@100
finetuned-LaBSE-tone	0.053	0.083	0.154	0.14	0.068
finetuned-LaBSE-entities	0.084	0.119	0.194	0.184	0.097
finetuned-LaBSE-overall	0.089	0.125	0.201	0.189	0.103
sentence-transformers/LaBSE	0.036	0.061	0.122	0.108	0.05
overall + entities	0.088	0.123	0.199	0.188	0.102
overall + entities - tone	0.09	0.126	0.207	0.194	0.102
random	0.006	0.006	0.005	0.005	0.005

Table 4.5: Precision scores for retrieving a review for the same product with a different rating.

the main reason the random baseline performs so poorly. Again, our model finetuned on the OVERALL dimension (finetuned-LaBSE-overall) outperforms the pre-trained LaBSE and random baseline. We also experimented with configuring the similarity search: We included the ENT dimension (overall + entities) and subtracted the TONE dimension (overall + entities - tone), which slightly improved the performance to 0.09. While the mean average precision (mAP) of 0.09 is a small number, it shows that on average almost every tenth retrieved review is for the same product with a different rating. For the random baseline, in contrast, only one of 166 retrieved reviews matches the search criteria.

These experiments show a promising result: configuring the similarity search by adding or subtracting certain dimensions can improve the results. This is an indication that our embeddings can a) be useful outside the domain of news texts and can b) be configured in an interpretable way.

4.3.2.2 Text Register Experiment

We conduct a document retrieval experiment using the Corpus of Online Registers of English dataset (Laippala et al., 2022). This dataset consists of almost 50,000 documents crawled from the unrestricted open Web, which were manually annotated with a taxonomy of eight main categories and tens of subcategories. For this experiment, we sample 1000 documents labeled with "Opinion Narrative" and 1000 documents labeled with "Informational Narrative".

Finetuned	Model Name	mAP
✗	LaBSE	0.574
✓	LaBSE-style	0.630
✓	LaBSE-tone	0.613
✓	LaBSE-narrative	0.539
✓	LaBSE-entities	0.529
✓	LaBSE-geography	0.529
✓	LaBSE-overall	0.534

Table 4.6: mAP scores for retrieving a narrative of the same category (opinion vs. informational).

With this experiment we test whether our models can distinguish between these two categories by simulating a typical document similarity search scenario: Given a document of a certain category, retrieve more documents of the same category. We hypothesize that these categories should be best separated by the STYLE or TONE dimension, as informational articles mostly use formal language, whereas opinion articles may use colloquial language. We do not expect the OVERALL, ENT or GEO dimension to be good discriminators, as both categories contain narratives about various topics.

The results shown in Table 4.6 confirm our hypothesis. A document retrieval system can find better results when utilizing a model finetuned on the STYLE dimension. While the difference in performance is not exceptional, we clearly observe that a document retrieval system returns better results when utilizing a finetuned model optimized for STYLE or TONE as opposed to using a model trained on paraphrases (pre-trained-LaBSE).

4.4 Structured Similarity Search

After looking into sentence/document-level similarity in the previous sections of this chapter, we now tackle the issue of directly retrieving specific spans significantly shorter than a sentence. In particular, we focus on retrieving named entities without using any named-entity recognition (NER) processing. To do so, we define suitable linguistic structures that allow to retrieve an entity span within a sentence of a potentially large document collection by finding the nearest embedding. This can be useful in the context of a retrieval scenario in which the ultimate goal is to enable users to rapidly find semantically similar entities in their datasets. It makes it possible to use the structured similarity search as an alternative to KWIC when querying named entities. Another usage of such a retrieval scenario is to speed up the manual annotation of text data, *e.g.* to create training data more efficiently.

4.4.1 Methodology

We approach the problem of semantic structured retrieval as a nearest neighbor problem, which can be described formally as follows:

Let $T := [t_1, \dots, t_n]$ be a dataset of d document with n tokens, where t_i represents a single token. For our retrieval experiments, we use datasets with class labels $y = [y_1, \dots, y_n]$, where

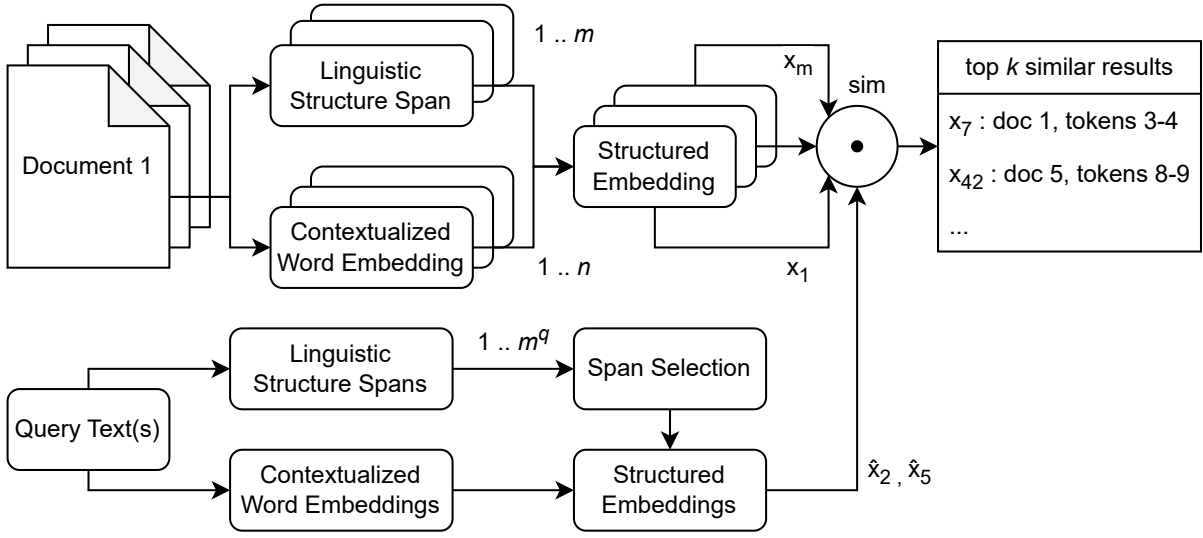


Figure 4.7: Overview of the indexing and retrieval process: During indexing, the text collection to be searched is split into linguistic structure spans where each span is transformed into a structured embedding using the CWEs of the contained tokens. At query time, the query is processed analogously although only a selected subset of the spans are used to produce the query embeddings. The top_k results are retrieved from the previously indexed collection allowing to find semantically similar text spans.

y_i is the label of token t_i . Documents are decomposed into a list of finer-grained, linguistic structure spans s such as tokens or multi-token units which we use as the basic unit of retrieval. For the whole dataset, a list of m linguistic structure spans $T \mapsto [s_1, \dots, s_m]$, with corresponding y_i labels $[y_1^1, \dots, y_m]$, is extracted by using a particular linguistic pattern. From the CWEs of the tokens belonging to any linguistic structure span s_i we generate a *structured embedding* $\mathbf{x}_i$. It represents a single feature vector extracted by a particular linguistic pattern.

The goal is to retrieve the k most relevant instances for a given query instance q and its extracted *structured embeddings* $q \mapsto \{\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_{m^q}\}$ of a target class c :

$$[r_1, \dots, r_k] := \underset{i \in \{1 \dots m\} \quad j \in \{1 \dots m^q\}}{\text{top}_k} \{ \arg \max \text{sim}(\hat{\mathbf{x}}_j, \mathbf{x}_i) \}$$

where top_k is defined as a function that selects the top k indices as an ordered list from the entire set of labeled instances regarding their maximum similarity score. The sim function is defined to be a similarity function for two vectors; we use *cosine similarity* in our experiments. Figure 4.7 illustrates the indexing and retrieval process. For evaluation, r_i is a correctly retrieved instance if $y_{r_i} = c$, i.e. if the class of the query instance matches the class of the retrieved instance. This allows us to determine the retrieval quality for any $k \geq 1$ with standard information retrieval measures.

Example usage in user-facing system A collection of sentences is indexed according to the above methodology. For simplicity, we use tokens as linguistic structures. Thus, the text collection is split into many spans of length one, so that every single token corresponds

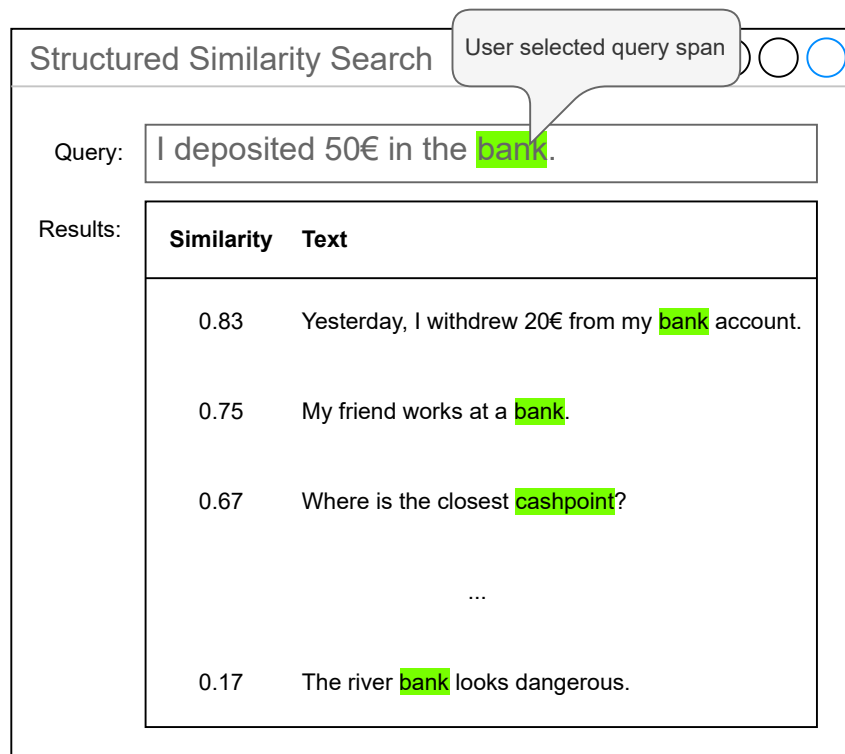


Figure 4.8: Mockup of a structured similarity search application. The user enters a search query text, selects specific spans as the query and the system provides the results ordered by similarity with the result span highlighted in green within their surrounding sentence.

to exactly one linguistic structure span. Then, CWEs are generated for every single token. Structured embeddings are created for each linguistic structure span by taking the CWE of its only token. Now, a user wants to perform a structured similarity search: The user writes a (short) text, the system produces both CWEs and linguistic structure spans; the user selects one or more spans as the relevant query parts (*Span Selection*). The system creates the final structured embeddings and performs a top_k similarity search against all structured embeddings indexed from the collection. The system returns the *k* best results to the user, *e.g.* each similar span highlighted within its surrounding sentence. A mockup of such an application is shown in Figure 4.8.

4.4.2 Linguistic Structures

We use the following two linguistic structures to find similar entity spans:

token: Each token of the dataset is considered a single item. Consequently, the unit of retrieval is always a single token.

SPS (same-PoS-span): In order to capture nouns and noun phrases, each sequence of tokens having the same PoS tag within a sentence is considered as one structure. Thus, the unit of retrieval is a variable-length span of one or more tokens.

For the linguistic pre-processing, *i.e.* tokenization and PoS tagging (see Section 3.2.3) we use *spaCy* (Honnibal et al., 2020). For CWEs based on RoBERTa (Liu et al., 2019), if a token comprises several word piece sub-word tokens, the corresponding embeddings are averaged to obtain a single vector for a lexical token. As SPS produces variable-length spans, the embeddings of all included tokens are averaged to form a single embedding per SPS structured embedding.

4.4.3 Experiments

For evaluation purposes, we test named entity retrieval on two datasets. We also compare with static word embeddings provided by word2Vec (Mikolov et al., 2013)¹ since our linguistic structures enable the composition of meaning due to the use of multiple tokens for a single structured representation. We investigate the retrieval performance using precision at k ($P@k$, $k = 1$ and $k = 5$) and mAP and use the static word2Vec embeddings and the contextualized RoBERTa embeddings. To perform the retrieval evaluation based on gold standard data, we use labeled datasets, which means each word or span is labeled with one specific target class. We use the standard train and test splits for indexing and querying as indicated by each dataset.

We additionally run a simple classification benchmark test using the same datasets. As a classification approach, we opted to use a k -nearest neighbors (k -NN) approach, which heavily relies on the retrieval performance and, thus, implicitly evaluates the retrieval performance. The k -NN approach groups and counts the class labels of the top k retrieved samples from the training set and uses the most prominent class label as a classification result. In case of ties, a random label of the most prominent class labels is chosen. Here, we report F_1 scores on the test sets and determine the hyperparameter k by using the validation set of the respective task benchmarks.

We use NER as a coarse-grained task since the label space of the multi-class classification problem is easily manageable. We evaluate the retrieval performance on the two common English benchmark datasets *CoNLL-2003* (Tjong Kim Sang and De Meulder, 2003) and *OntoNotes Release 5.0* (Weischedel et al., 2013).²

For retrieval, we only use structure spans consisting of entity-labeled tokens, *i.e.* excluding the ‘other’ class as queries — with the goal to find more structures having the same label as the query. For NER, searching for non-entities, and including their scores, would only increase the reported performance, because the majority of labels are actually ‘other’. Thus, analogously to standard NER classification evaluation, our retrieval scores do not include the *other* class, which is a majority class label.

Both structures explained in Section 4.4.2 are tested. An issue arises when retrieving arbitrary token spans instead of whole sentences because the unit of retrieval is some linguistic structure that does not necessarily map perfectly to an entity span. Since there is no proper solution to this issue, we validate the appropriateness of our linguistic structures used for retrieval via named-entity classification. The classification scores allow interpretation and connection to state of the art (SotA) results, but we note that those results are only for anecdotal purposes and cannot be properly compared to SotA systems because of the simplicity and

¹<https://code.google.com/archive/p/word2vec/>

²We apply the split proposed by Pradhan et al. (2013) for OntoNotes as there is no official dataset split.

Data	Structure		
	token	SPS	
CoNLL-2003 (word2vec)	37.1	38.9	mAP1K
	71.3	79.8	P@1
	64.5	70.3	P@5
CoNLL-2003 (RoBERTa)	48.0	48.0	mAP1K
	87.3	87.2	P@1
	78.1	79.3	P@5
OntoNotes-v5 (word2vec)	26.6	29.6	mAP1K
	49.7	50.5	P@1
	38.7	44.9	P@5
OntoNotes-v5 (RoBERTa)	38.4	36.0	mAP1K
	75.7	75.3	P@1
	64.4	64.5	P@5

Table 4.7: NER retrieval results. We use the mAP estimate of the top 1K nearest neighbors.

different objective of our approach. To perform classification, we use the k -NN approach based on the retrieval result set for each queried linguistic structure in the test set.

We further performed experiments with varying background corpus sizes in order to simulate how little data needs to be manually annotated for our *more-like-this* approach to work reasonably.

4.4.4 Results

For discussion, we focus on P@1 scores because we believe this is the most important metric for practical applicability. As expected, we observe significantly better performance using CWEs as compared to static word embeddings across all settings. However, our goal is not to compare these two types of embeddings, but to evaluate if aggregation of embeddings along linguistically informed lexical structures allows to directly retrieve spans shorter than a sentence regardless of the type of embedding.

4.4.4.1 Named-Entity Retrieval

Table 4.7 shows the retrieval results for the CoNLL-2003 and OntoNotes v5 datasets. The retrieval performances of the two structures differ depending on the type of word embedding. We see a difference of 10–15 F_1 points for each dataset and aggregation strategy. With static word embeddings, the SPS structure shows improved performance compared to the token structure. A likely explanation is that averaging vectors of neighboring words inherently creates a kind of composite embedding that is unique for the combination of words. This is supported by the observation that for CWEs, there is only a minor difference between both linguistic structures. While the performance difference decreases with the size of k for the CoNLL dataset, it increases for the OntoNotes dataset. In any case, proper CWEs perform

Dataset	Embedding	Structure	k	F_1
CoNLL-2003	word2vec	token	3	64.5
CoNLL-2003	RoBERTa	token	1	79.4
CoNLL-2003	word2vec	SPS	3	73.5
CoNLL-2003	RoBERTa	SPS	3	79.6
OntoNotes-v5	word2vec	token	25	44.9
OntoNotes-v5	RoBERTa	token	11	65.6
OntoNotes-v5	word2vec	SPS	16	52.3
OntoNotes-v5	RoBERTa	SPS	9	65.9

Table 4.8: Classification results using k -NN for word-level tasks (upper part) and sentence-level tasks (lower part). k refers to the best identified k on the validation set.

significantly better than using averaged word2vec embeddings for multi-word expressions. For small k , SPS is marginally better while token outperforms SPS on the mAP-1K metric on the OntoNotes dataset.

4.4.4.2 Named-Entity Classification

The full classification results for CoNLL-2003 and OntoNotes-v5 are shown in Table 4.8. The best k was identified using a held-out validation set. The final F_1 score was produced on the held-out test set. Overall, the picture is very similar to retrieval. There is only a minor difference between both structures when using contextual embeddings. Interestingly, the best classification setups do not correlate with the precision at k scores in the retrieval setup, but rather the mAP scores. While the classification with the k -NN approach does not reach SotA performance (our best setting = 79.4 vs. 94.3 F_1 by Yamada et al. (2020) on CoNLL-2003 and 66.5 vs. 92.1 F_1 by Li et al. (2020a) on OntoNotes v5), the approach still achieves considerable results over a standard baseline. The classification scores show that both linguistic structures are generally useful to retrieve named entities of the same type. The significantly shorter training and prediction times of k NN-classification compared to fine-tuning transformers make it an appealing approach in some scenarios (e.g. interactive usage) despite the significantly lower performance.

4.4.4.3 Rapid Nearest Neighbor Classification

Furthermore, k -NN can be used in few-shot classification scenarios. In a further experiment, we test the performance of the classifier with increasing dataset size, where we randomly select training sentences for indexing. Figure 4.9 shows the classification performance with increasing number of training sentences. We only show the best setup that was identified from Table 4.8. With just 100 labeled sentences, the k -NN approach achieves roughly 80% of the full CoNLL dataset performance (62.7 F_1). Due to OntoNotes larger size, between 1000 and 3000 labeled sentences are needed to obtain a similar relative performance of 80%. We can see that as few as 3,000 sentences are sufficient to reach a decent performance level that only slightly increases with more training data. The quickly saturating performance level on this logarithmic scale shows the large potential to use a *more-like-this* approach to speed up the

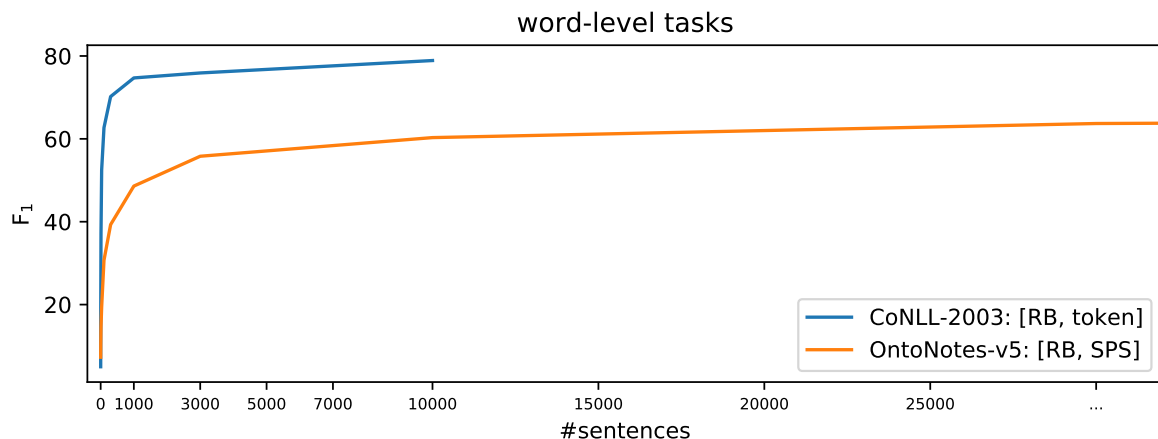


Figure 4.9: k -NN performance for increasing training dataset sizes for the word-level task NER. Only the best settings and their scores are illustrated. Figure 4 (a) from Remus et al. (2022).

manual annotation of a new sequence tagging dataset. We continue this thought and devise an elaborate method to scale manual annotations to a large corpus in Chapter 7.

4.5 Conclusion

In this chapter, we tackled the lack of semantic similarity search usage in interdisciplinary DH or CSS contexts due to 1. low explainability resp. interpretability and 2. missing support for a semantic KWIC-like search.

Regarding the first issue, we introduced a novel semantic search configurable by the user to focus on different dimensions. Since datasets for this are scarce in the NLP community, we trained our models on the only available dataset from the SemEval 2022 Shared Task 8 (Chen et al., 2022). Based on this single dataset, we created multiple models focussing on the different similarity dimensions annotated in the dataset. While our MTL model performs poorly as compared to human judgements, the individually fine-tuned models perform well. We used the models to construct a configurable semantic similarity search system and showed the applicability on experiments emulating the usage of such a system on two out-of-domain datasets.

Our models achieve state-of-the-art performance on in-domain data for various dimensions and have successfully been applied to out-of-domain data using some of these dimensions. The trained models are to some extent applicable to other datasets for the STYLE, TONE, ENT and OVERALL dimensions. Further, we found that the foundation models pre-trained capabilities were not removed by our domain-specific dimension fine-tuning.

Limitations of the Dimension-based, Configurable Similarity Search While our model performs well on a variety of tasks there are multiple major limitations to be discussed. First, we did not test the NAR, TIME and GEO dimensions because we found no suitable annotated dataset to experiment on. Thus, the applicability of these dimensions beyond the SemEval corpus is unknown. For the case of the NAR dimension, as it is strongly correlated with the

OVERALL dimension, we assume that it will behave similarly. We have no reason to believe that the GEO dimension will present a challenge but suspect that the TIME dimension may not be easily captured by our models as multiple participants in the original shared task implemented special handling for time and dates. In terms of cross-domain performance, there is still room for more exploration, although we experimented on out-of-domain datasets. Our approach will have to be tested with human users in the future, as our application experiments were a bit contrived and not driven by real-world use cases — they were driven by availability of annotated datasets. A major limitation of our work is that we only ever consider the first 512 subword-tokens of an article, which is a limitation in the models’ pre-training which, in turn, is motivated by memory constraints. In news articles, this is likely not a significant problem due to the inverted-pyramid property (Pöttker, 2003) but is something we will seek to address in future work.

We see the weighted combination of the different dimensions’ similarities as suitable base to build systems providing a human-interpretable, configurable similarity search. First, search systems using our models may help in building a more interpretable notion of similarity by providing the similarity scores for different dimensions between two (or more) documents. Second, a dimension-empowered search system enables users to balance the aspects of similarity so that the search results better reflect what dimensions are important to the user for a specific query. Such a search system allows answering requests such as: “show me articles on the same topic but from different perspectives.” In the future, we aim to implement such a system in an easily usable software platform, enabling DH and CSS researchers amongst others to employ a more interpretable, configurable semantic similarity search for their analyses.

Regarding the second issue of the missing support for a semantic KWIC-like search, we presented an approach using structured similarity to find semantically similar units for variable-length entity spans. We used linguistically informed structures to select span candidates and create embeddings. We evaluated the feasibility of our approach in an information retrieval scenario and showed that it is possible to retrieve variable-length entity spans using linguistic structures on top of CWEs without any fine-tuning. Further, we tested k -NN classification on the structured embeddings and found that only few labeled data is needed to obtain decent performance. Beside offering a search for sub-sentence spans (such as the widely used KWIC method), we see an additional use case for our approach by speeding up manual annotation of corpora in CSS or DH contexts. We develop an elaborate method to scale manual annotations to a large corpus in Chapter 7.

Limitations of the Structured Similarity Search While our approach performs well on NER, the method can be applied to other tasks as well given appropriate structures (see Remus et al. (2022)). However, in our experiments we focused only on named-entities as these were deemed central for an application in interdisciplinary DH settings by numerous projects partners. Having successfully verified the applicability of a structured similarity search in a retrieval scenario is only the first part towards a usable system, which we plan to address in the future.

Having presented both dimension-based configurable similarity search and a structured similarity search, the natural question arises whether it is possible to combine both into a single, unified similarity search providing sub-sentence results from the structured search in a configurable manner from the dimension-based search. While theoretically possible, such an approach would require annotated data measuring each type of similarity between two text

spans for a large variety of potential structures. Since that is a massive annotation effort, we did not pursue this direction further.

Overall, the work presented in this chapter makes a large step towards a widely usable semantic similarity search for DH and CSS by providing models and methods for overcoming two common issues.

In the next chapter, we tackle another NLP task applicable in various DH and CSS contexts: Identifying and attributing quotations.

5 Quotation Detection and Attribution

Abstract Extracting who says what to whom is a valuable information for a wide variety of analyses in the Digital Humanities (DH) and Computational Social Science (CSS). In this chapter, we tackle the issue of quotation detection and attribution in German news articles. We introduce the rather complex task in detail, review related work, and create a new manually-annotated training dataset. Then, we develop three different approaches: A rule-based QUIiTE software not relying on any training data, an adaption of the existing machine learning (ML)-based Citron (Newell et al., 2018) software, and our novel neural ML model Quotect. We evaluate the three models on our dataset and find Quotect to achieve highly superior quotation detection and attribution. Finally, we conclude our findings, discuss ethical aspects and limitations, and make the case for combining our model with coreference resolution.

This chapter is (partly) based on the following publication(s):

Fynn Petersen-Frey and Chris Biemann (2024a). “Dataset of Quotation Attribution in German News Articles”. In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. Torino, Italia: ELRA and ICCL, pp. 4412–4422. URL: <https://aclanthology.org/2024.lrec-main.394>.

Fynn Petersen-Frey and Chris Biemann (2024b). “Fine-grained Quotation Detection and Attribution in German News Articles”. In: *Proceedings of the 20th Conference on Natural Language Processing (KONVENS 2024)*. Ed. by Pedro Henrique Luz de Araujo, Andreas Baumann, Dagmar Gromann, Brigitte Krenn, Benjamin Roth, and Michael Wiegand. Vienna, Austria: Association for Computational Linguistics, pp. 196–208. URL: <https://aclanthology.org/2024.konvens-main.22/>.

5.1	Introduction	68
5.1.1	Task Description	68
5.1.2	Use Cases	69
5.1.3	Chapter Overview	70
5.2	Related Work	70
5.2.1	Quotation Detection and Attribution	70
5.2.2	Semantic Role Labeling	72
5.2.3	Structured Generation	72
5.3	News Article Dataset with Quotation Annotations	72
5.3.1	Annotation Schema	74
5.3.2	Dataset Creation	77
5.3.3	Quantitative Analysis	81
5.4	Systems	84
5.4.1	QUIiTE: A rule-based System	84
5.4.2	Citron	86
5.4.3	Quotect: Fine-grained Quotation Detection and Attribution	86
5.5	Experiments	93
5.5.1	Evaluation Metrics	93

5.5.2	Data	94
5.5.3	Quotect Model Variants	94
5.5.4	Training Details	94
5.5.5	Main Results	95
5.5.6	QUiTE and Citron Ablation Study	98
5.5.7	Quotect Model Variants Ablation Study	99
5.5.8	Inference Runtime Efficiency	101
5.5.9	Error Analysis	102
5.6	Conclusion	102
5.6.1	Ethical Considerations	103
5.6.2	Limitations	104
5.6.3	The Case for Coreference Resolution	104

5.1 Introduction

Scientists and journalists who want to perform manual analyses on today’s abundant textual data require specific solutions to reduce it to manageable amount. In particular, extracting who says what to whom is a crucial part in analyzing human communication. A multitude of journalistic research questions could be more easily answered given this information for a specified collection of articles, *e.g.* how the discourse changes over time, which quotations are reproduced by which media *etc.*

The field of natural language processing (NLP) has seen huge improvements with the introduction of transformers, often making a certain machine-based automation work well enough for users for the first time. Yet, for German news articles there exists no transformer-based system to extract who said what to whom. To remedy this, we develop such a system from the ground up by making the following two key contributions: (1) We create a new, freely available, creative-commons-licensed dataset for quotation attribution in German news articles based on Wikinews (see Section 5.3). The dataset provides curated, high-quality annotations across almost 1000 documents (250,000 tokens) in a fine-grained annotation schema enabling various downstream uses for the dataset. (2) We build a new transformer-based system (see Section 5.4.3) trained on that dataset to properly solve the problem of extracting who said what to whom.

5.1.1 Task Description

News articles typically contain factual information and include quite a number of quotations, both direct and in various forms of paraphrases. These quotations are often used as evidence and build the foundation of an entire article. This makes news articles especially relevant in the context of automatic quotation extraction. However, quotations also occur in many other texts such as literary works where they are of interest to analyze as well. Automatic ways to find, highlight, extract quotations with their speakers are highly valuable to DH & CSS researchers (see the example use cases in the following subsection).

The task of quotation detection and attribution deals with identifying quotation spans (*Quote*) in a document together with their associated role spans such as the *Speaker*, *Cue*,

Addressee or *Frame* (context). The result is a list of tuples per document. While early, simple variants of the task only considered *Direct* quotations spanning at most a single sentence, the task’s complexity has increased over time (see Section 5.2.1). Our work also features a significantly more complex task that requires a fine-grained quotation detection of five quotation types and attribution of up to four roles per quotation.

Example 5.1.1 shows a human-friendly representation of an annotated text passage created from a NASA blog article¹. It contains two quotations (*Direct* and *Indirect*) uttered by the same *Speaker*. The *Indirect* quotation is further invoked by a *Cue* word “said” within a *Frame* in the same sentence.

Example 5.1.1 (Annotated text passage): *The NASA Jet Propulsion Laboratory released a statement on their Mars Helicopter. MiMi Aung said this was the first time Ingenuity survived*

the night by itself while being exposed to Mars’ atmosphere directly.
“But we now have confirmation that we have the right insulation, the right heaters, and enough
energy in its battery to survive the cold night, which is a big win for the team. We’re excited to
continue to prepare Ingenuity for its first flight test.”

A quotation with all its associated roles is called a *Quote* group. The integers behind each span type indicate which group that span belongs to. In case of the *Speaker* “MiMi Aung”, it belongs to both the first and second group. Example 5.1.2 shows the machine-readable annotations as token offsets (token start – token end) in the original text passage.

Example 5.1.2 (Offset annotations):

Group 1: Speaker(14–15), Frame(14–16), Cue(16–16), Indirect(17–34)

Group 2: Speaker(14–15), Direct(36–84)

In our fine-grained task variant and dataset, a single quotation can span multiple sentences, each quotation can be associated with zero to four roles, *Quote* groups can be nested inside each other, the same annotation span can belong to multiple groups and annotation spans can be discontinuous. We solve the full task with all its challenges and predict *Direct*, *Indirect*, *Reported*, *Free Indirect* and *Indirect/Free Indirect* quotations together with the *Speaker*, *Cue*, *Addressee* and *Frame* roles.

5.1.2 Use Cases

In this section, we outline envisioned use cases with project partners from DH and CSS. We demonstrate how our work can help researchers work on their research questions. Reliably extracting a list of *Quote* groups from a large document collection allows researchers or journalists to quickly analyze the quotations contained in their data of interest. As the system not only provides the *Speaker* and *Quote* but also the *Cue* and *Frame* to automatically detect negations, it is feasible to aggregate quotations by their *Cue* word to collect the desired samples or produce statistics on these aspects. The type of quotation and existence of certain roles can

¹<https://www.nasa.gov/solar-system/nasas-mars-helicopter-survives-first-cold-martian-night-on-its-own/>

be used to filter and/or aggregate quotations as needed. For example, social climate science researchers can compare statements after grouping the speakers into politicians, environmental activists, corporate representatives and other public figures (this is possible after performing coreference resolution and entity linking on the documents). Thereby, researchers can both have a quantitative view on the quotations in their data of interest and qualitatively analyze individual quotations and/or speakers by filtering all detected quotations for certain aspects.

The fine-grained annotations also enable corpus comparisons using quotation statistics such as the distribution of quotation types, fraction of quotations with or without a *Speaker* or *Addressee* or general frequency of quotations in the corpus. This can be used to compare different news outlets based on the general frequency resp. fraction of text being a quotation as well as the attributes of quotations used: Type of quotation (e.g. *Direct* versus some *Indirect* form), presence versus absence of a *Speaker* etc. After collecting news articles on the same topic during the same timeframe for various media outlets, researchers can quantitatively compare the news outlets and analyze whether this correlates with news outlet metadata such as reach, geographical location, position in the political spectrum. Further, it is possible to check if individual quotations occur in multiple news outlets or only once – these cases could be candidates for a manual verification or otherwise of interest.

5.1.3 Chapter Overview

In the remainder of this chapter, we first review related work regarding quotation detection and attribution, the similar task of semantic role labeling (SRL), and structured generation for our ML model. Second, we describe our new, free dataset for the task with the detailed annotation schema, creation of the dataset including data collection and the annotation process, and a quantitative analysis of the final dataset. Third, we describe three systems that we apply on the dataset: Our rule-based QUI TE system, the existing ML-based Citron (Newell et al., 2018) software, and our novel, neural Quotect model. Forth, we present our experiments including the setup, main results, ablation studies, runtime analysis, and an extensive error analysis. Finally, we conclude our findings, discuss ethical aspects and our models’ limitations, and make the case for coreference resolution to further boost the applicability of our models.

5.2 Related Work

We first review previous work on quotation detection and attribution for English and German. Then, we review approaches to semantic role labeling (SRL) and structured generation for our ML model developed in Section 5.4.3.

5.2.1 Quotation Detection and Attribution

The task of quotation detection and attribution to a speaker has been tackled by numerous approaches, usually with the goal to extract information from the data such as news articles. Earlier approaches were purely rule-based and only dealt with quotation detection. More recent works used data-driven methods (often based on the PARC dataset by Pareti (2012)) to detect quotations and their respective speakers.

The NewsExplorer system (Pouliquen et al., 2007) is the first system to tackle the task of *Direct* quotation detection and attribution to a *Speaker*. It extracted *Direct* quotations with

three parts using word lists and rules: speaker name (*Speaker*), reporting verb (*Cue*) and the quotation (*Quote*). Krestel et al. (2008) also detected *Indirect* quotations using a rule-based reporting verb and reported speech finder. O’Keefe et al. (2012) created a new labeled dataset and reformulated *Direct* quotation detection and attribution as a sequence labeling task. Pareti et al. (2013) focus on the more challenging *Indirect* quotations by training classifiers for the *Cue*, *Speaker* and *Quote*. Almeida et al. (2014) introduced a model that jointly solves the problems of quotation attribution and coreference resolution. Scheible et al. (2016) experiment with new model architectures to perform quotation attribution. Newell et al. (2018) created the Citron software that implements an improved variant of the approach by Pareti et al. (2013) using trainable content resolver and source resolver. Zhang and Liu (2022) focus only on *Direct* quotations and test multiple sequence labeling methods including three neural models.

There is still a strong focus on resources for direct quotations such as the software by Pouliquen et al. (2007) or the datasets by O’Keefe et al. (2012) and Zhang and Liu (2022) while far fewer resources also provide indirect quotations as they are not as easily extracted automatically.

A similar task is speaker attribution in literary works. As the literary domain differs from the news domain *e.g.* in the author perspective, in the type of quotations and the focus on characters as implicit or explicit speakers, the field of computational literary studies has seen numerous studies dealing with speaker attribution in literary works (Elson et al., 2010; He et al., 2013; Muzny et al., 2017).

While quotation detection and attribution in English news has been addressed by many works, less work and resources have been created for German. Akbik and Schenck (2013) created QuoteMine, a repository of quotes with the corresponding speakers extracted automatically from German news articles. Their rule-based system performs named-entity recognition (NER) and resolves proper noun coreferences to detect speakers, extracts text in quotation marks for *Direct* quotes, and finally attributes quotes to speakers with a set of manually crafted heuristics. The closed-source system provided a web interface to query quotes of certain persons or quotes containing a specific keyword. Bögel and Gertz (2015) created a rule-based system using dependency trees to extract and attribute statements (source, cue, content, target, timestamp) from German news articles. They consider both *Direct* and *Indirect* quotation. Their system is not made available to use or reproduce experiments. Papay and Padó (2019) created a corpus-agnostic, neural quotation detection model that can detect *Direct* and *Indirect* quotations but does not perform any attribution to *Cue* or *Speaker*. The Zitatsuchmaschine² (Dallabetta et al., 2024) is a closed-source search engine for quotes that were automatically extracted from German online news articles. It detects *Direct* quotes across multiple news outlets and links these to a single speaker via entity linking.

For historical German text, especially in the literary domain, Brunner (2015), Krug et al. (2018), and Brunner et al. (2019, 2020a) have created a number of resources. The DROC corpus (Krug et al., 2018) consists of 90 fragments of German novels and includes about 2000 annotated direct quotes and annotations for speakers and addressees. The Redewiedergabe corpus (Brunner et al., 2020a) extends this work by creating a historical corpus (mostly literary domain, but also some news articles) with annotations for speech, thought and writing. Tu et al. (2019) experimented with a rule-based approach to identify speech without quotation marks in

²<https://zitatsuchmaschine.informatik.hu-berlin.de/>

literary works. Brunner et al. (2020b) trained simple sequence taggers on the Redewiedergabe corpus to individually detect *Direct*, *Indirect*, *Reported* and *Free Indirect* quotations without any roles or attribution. They neither detect roles nor perform attribution.

5.2.2 Semantic Role Labeling

Semantic role labeling (SRL) deals with discovering the predicate-argument structure of a sentence by identifying predicates, arguments and their semantic relationships. The majority of approaches to SRL rely on information that is only available in the specific SRL datasets (such as gold predicates and syntactic dependency trees). This makes them impractical for real applications that require full SRL on raw text data. Cai et al. (2018) present the first syntax-agnostic method for full end-to-end SRL. They perform predicate disambiguation and argument labeling in a single model by exhaustively scoring all predicate-argument relations and their semantic roles. Decoding is performed in a two-step process by first identifying predicates. Then, for each predicate its arguments are identified and labeled. Li et al. (2019) and Li et al. (2020b) improve upon this by removing the two-step process and jointly processing predicates and arguments in a unified fashion. Zhou et al. (2022) improve the latter to full end-to-end SRL without pre-identified predicates. Daza and Frank (2018) frame SRL as a sequence-to-sequence process using constrained generation with a copy mechanism. While their model can solve SRL on English, further improvements are needed to make it competitive. Fernández-González (2023) propose the first transition-based, syntax-agnostic, end-to-end SRL approach achieving state-of-the-art performance. It is based on Pointer Networks and can process one sentence in a single left-to-right pass.

5.2.3 Structured Generation

Conditional language modeling has become a useful technique to tackle structured prediction tasks using pre-trained language models (PLMs). The target structure is flattened into a sequence and a conditional language model is trained to predict it. Paolini et al. (2021) solve structured prediction language tasks (e.g. joint entity and relation extraction and SRL) by performing translation between augmented natural languages. Liu et al. (2022) argue that flattening structured information into a sequence leads to inferior performance. They model structures as action sequences allowing in-structure dependencies to be learned and achieve a new state-of-the-art on NER, relation extraction and coreference resolution. Geng et al. (2023) suggest grammar-constrained decoding on language models without fine-tuning can solve many structured NLP tasks and show this for information extraction, entity disambiguation and constituency parsing. Zhang et al. (2023) show that task-specific models are not necessary for state-of-the-art coreference resolution performance with the right choice of sequence representations. They finetune a sequence-to-sequence transformer to translate a plain input document to a sequence encoding the coreference annotation.

5.3 News Article Dataset with Quotation Annotations

Although the field of NLP has made huge leaps forward with the introduction of transformers, there was no previous transformer-based system to extract who said what to whom for modern

Annotation	Short Description	Count	Avg. Len.
<i>Quote</i>	the quotation uttered by the <i>Speaker</i> , fine-grained labels in Table 5.2	4182	16.69
<i>Speaker</i>	entity in the text that utters the quotation	3908	3.53
<i>Cue</i>	words that are part of a <i>Frame</i> and signal a <i>Quote</i> construction	2929	1.57
<i>Frame</i>	part of a sentence including <i>Cue</i> & <i>Speaker</i> , but not the quotation	3038	8.95
<i>Addressee</i>	entity in the text that the quotation is directed at	337	2.72

Table 5.1: Overview of quotation attribution spans

German news articles — mainly because there exists no annotated dataset for this task. To create a high-quality ML system annotated resources are required to train models that can learn to predict quotations with their speakers and addressees in natural language articles.

In this section, we present our work of creating a free, open-access, creative-commons-licensed dataset for quotation attribution in German news articles.³ The dataset consists of almost 1000 manually annotated articles from the German Wikinews website⁴. In total, these annotated articles contain approximately 250,000 tokens. We manually annotated and curated *Quotes* in different forms of speech such as *Direct*, *Indirect*, *Free Indirect*, *Indirect/Free Indirect*, *Reported* together with the corresponding *Frame*, *Speaker*, *Cue* and *Addressee*.

An overview of the span annotation classes can be found in Table 5.1. This includes short descriptions, number of occurrences in the data and their average length. Table 5.2 gives an overview of quote types including short descriptions, the number of occurrences and average length. In addition, we provide a number of annotated sentences in Examples 5.3.1–5.3.5 to get a quick intuition on the dataset and its annotations. These examples are modeled after cases from the curated dataset. We shortened or changed the content as needed to be presentable in this text while keeping the structure and grammatical phenomenon as it was.

Example 5.3.1 (*Direct*):

Zitat von Merkel: „Wir schaffen das.“
Cue Speaker Direct
Frame

Quote from Merkel: „We can do this.“
Cue Speaker Direct
Frame

Example 5.3.2 (*Indirect*):

Der Nachrichtenagentur sagte er, dass man eine Lösung finden werde.
Addressee Cue Speaker Indirect
Frame

He told the news agency that a solution would be found.
Speaker Cue Addressee Indirect
Frame

³Available at <https://github.com/uhh-lt/german-news-quotation-attribution-2024>

⁴URL: <https://de.wikinews.org>

Type	Short Description	Count	Avg. Len.
<i>Direct</i>	actual words of an utterance, usually in quotation marks	873	17.54
<i>Indirect</i>	content-wise equivalent utterance using different words, usually part of a sentence together with a <i>Frame</i>	2250	14.71
<i>Reported</i>	report of a speech action, possibly far from the original quote, usually a full sentence, no <i>Frame</i>	454	18.01
<i>Free Indirect</i>	mix of article author & actual speaker, typically construct with "sollen" (shall) or "müssen" (must), full sentence	171	20.42
<i>Indirect/Free Indirect</i>	content-wise equivalent utterance written in conjunctive mood, full sentence	434	22.33

Table 5.2: Overview of the quotation types

Example 5.3.3 (*Reported*):

Die Firma forderte eine schnellere Entscheidung.

Speaker

Reported

The company demanded a quicker decision.

Speaker

Reported

Example 5.3.4 (*Free Indirect*):

Ein Sprecher stellte gestern die neuen Ziele vor. Es soll mehr Geld in die Bildung fließen.

Speaker

Free Indirect

A spokesman presented the new goals yesterday. More money is to flow into education.

Speaker

Free Indirect

Example 5.3.5 (*Indirect/Free Indirect*):

Ein Passant schilderte die Situation. Die Polizei habe den Bereich großräumig abgeriegelt.

Speaker

Indirect/Free Indirect

A passerby described the situation. The police had cordoned off the area over a wide area.

Speaker

Indirect/Free Indirect

In the following, we describe our annotation schema, the creation of our dataset, and perform a quantitative analysis of our final dataset to deduct which traits are required in automatic systems to be able to perform well.

5.3.1 Annotation Schema

The annotation schema is inspired primarily by the Redewiedergabe project (Brunner et al., 2020a) and also by the work of Bögel and Gertz (2015). We annotate five different (possibly discontinuous) spans: Beside the actual quotation annotated as *Quote*, spans of *Speaker*, *Cue*,

Frame and *Addressee* (not part of the Redewiedergabe project) are annotated as optional roles for each *Quote*. For a *Quote* two additional dimensions are coded: Five different types of speech (*Direct*, *Indirect*, *Reported*, *Free Indirect*, *Indirect/Free Indirect*) as well as six media (*Speech*, *Thought*, *Writing*, and not part of the Redewiedergabe project *Speech/Thought*, *Speech/Writing*, *Writing/Thought*). This produces 30 different combinations of quotations. Examples 5.3.1–5.3.5 provide short sentences showing the annotations. While we reused some class names from the Redewiedergabe project and tried to align our classes, we modified definitions from the Redewiedergabe annotation guidelines or created new definitions suited for news articles and nested quotations.

Refer to our own annotations guidelines (in German) accompanying the dataset⁵ for details in what situations which annotation is used. In the next sections, we briefly define the roles, types and media.

5.3.1.1 Roles

A quotation in itself is of little value when it is not known who said it or in what context the statement was made. Thus, we provide roles as additional spans that are linked to one or more quotations. The *Speaker* identifies who said something, the *Cue* describes how it was uttered (possibly negating/adversary!) and the *Frame* provides context so that a quotation is not free-floating.

Speaker The speaker is the linguistic phrase in the text that utters one or more *Quotes*. This is typically a personal pronoun, named entity or a noun phrase subject in a sentence. Explanatory relative clauses as well as content clauses (clauses with *dass* (that) or *ob* (whether) making an attribution to a subject) are not annotated. However, noun phrase modifiers (explanatory attributions of subjects) are annotated (e.g. der zufälligerweise anwesende Doktor (the doctor who happens to be present) is the full *Speaker*. Usually, the speaker of *Direct* and *Indirect* speech is located within the associated frame. For *Reported*, the speaker can be found within the quotation span. The *Speaker* of *Free Indirect* and *Indirect/Free Indirect* is outside the respective quotation span.

Cue A *Cue* consists of signal words in a frame that announce a *Direct* or *Indirect* speech. These are usually verbs. However, they can also be specified expressions (e.g. : laut, nach, so, zufolge (according to). The *Cue* span can also be split within a frame (typically for German verb pre- or suffixes). Besides the reflexive pronoun, a verb can also include other parts of speech such as prepositions, adverbs, nouns and adjectives which distinguish the verb from similar verbs; e.g. : von ... die Rede sein (talk of ...), für wahrscheinlich halten (consider likely).

Addressee The *Addressee* is the linguistic phrase in the text that a quotation is directed at. It is typically found within a *Frame* or within the quotation span in case of *Reported*.

Frame The part of a sentence that is outside the quotation and contains *Cue*, *Speaker* and possibly *Addressee*. The frame provides context for the quotation. It can be at the beginning, in the middle or at the end of a sentence. It is also possible to split the *Frame* within a sentence if it is interrupted by the quotation. The *Frame* in *Indirect* and *Direct* speech is usually a clause, annotated with its comma or colon, which separates the quotation from the *Frame*.

⁵<https://github.com/uhh-lt/german-news-quotation-attribution-2024>

5.3.1.2 Quotation Types

Quotations come in various forms and shapes. These differ in their level of truthfulness of the reproduction to the original utterance. To account for this, we marked a span of text not only as a *Quote* but also labeled it according to the five classes described in the following paragraphs. This additional information per quotation allows to use the dataset for downstream tasks that are only interested in a specific type of quotation. When further processing extracted quotations, systems can consider the reproduction truthfulness in their methods and differentiate e.g. between a *Direct* quotation and only vaguely related *Reported* speech.

Direct The *Direct* label is used for verbatim reproductions of quotations that usually occur enclosed in quotation marks. Typically, it either directly follows an introductory *Frame* or the associated *Frame* immediately follows the quotation. *Frame* and *Direct* speech can be separated either by colons or commas that belong to the *Frame*. In other cases, fragments of *Direct* quotations are integrated into a sentence and cannot stand on their own. Then, the *Direct* speech is nested in a longer quotation of any of the remaining four quotation types.

Indirect A quotation is labeled as *Indirect* whenever the author of a text indicates in an associated *Frame* that the utterances of another person are reproduced as a paraphrase, not verbatim. It is the only type of *Quote* that always requires a *Frame*. The usual type of *Indirect* speech is a partial sentence that, together with a *Cue* in the *Frame*, forms the complete sentence. In a special case, the *Cue* is a single word reference (e.g. *wonach*, *demnach*, *danach* whereupon, thus) to a *Speaker* in the previous sentence.

Reported A *Reported* quotation is a summary report of a statement made by another person that is reproduced in a free manner possibly far from the original statement. Because a reporting style is common in news texts, *Reported* speech is annotated only in cases where, first, there is a clearly identifiable *Speaker* within the quotation span, and second, the quotation contains information uttered by the *Speaker* – not only a description of an action.

Free Indirect A quotation is labeled *Free Indirect* when statements, writings or thoughts of a person are reproduced who is not the article author, but the quotation is nevertheless written from the author’s perspective. Mostly formulations with *sollen* (shall) and *müssen* (must) that reflect foreign thoughts, statements or writings are considered. A *Free Indirect* quotation is usually a complete sentence that is not enclosed by quotation marks, nor does it have a *Frame* in the same sentence. However, it has a *Speaker* that is outside the quotation span and thus also outside the sentence.

Indirect/Free Indirect This class is used for those forms of speech reproduction which, without an introductory *Frame*, reproduce statements of another entity in a sentence in the subjunctive mood, but not verbatim. An *Indirect/Free Indirect* quotation occurs only when any of the other four types of quotation occur in the preceding or succeeding sentence that also provide a *Speaker*.

5.3.1.3 Quote Media

The media of speech reproduction indicate whether a quotation is a *Speech*, *Thought* or *Writing* action. These three media are only annotated if it can be clearly determined which particular

medium was used in a quotation. If the media cannot be clearly determined or two different media apply at the same time, the mixed forms are chosen.

Speech A *Quote* is labeled *Speaker* when the reproduced utterance was originally oral. This medium of speech reproduction is often recognizable in newspaper texts by *Cue* verbs that are unambiguous for a spoken reproduction; e.g. : *sagen* (say), *sprechen* (speak). But also the *Speaker* can give clarity about a spoken utterance; e.g. *der Sprecher* (the speaker).

Thought The *Thought* class marks a reproduced cognitive process where the statement originally occurred mentally. Consequently, *Thought* is rarely found in news articles as the author cannot reproduce the thoughts of another person.

Writing This class labels a reproduced writing process or a written form of language, i.e. the reproduced quotation originally took place in writing. Similar to *Speech*, the *Cue* or *Speaker* can be indicators.

Speech/Thought This mixed medium is used to label a person's oral statement in which they have expressed their thoughts.

Speech/Writing This mixed medium marks a *Quote* a) when it's uncertain whether the original quote comes from a written or oral source or b) if a quotation is made as a combination of texts as well as oral statements.

Writing/Thought This mixed medium is chosen when a person's writing is cited in which he or she has reproduced his or her thoughts. Our dataset of news articles has no instances of this class.

5.3.2 Dataset Creation

Having described the annotation schema, we provide details on the source data, its pre-processing, the annotation process, the inter-annotator agreement and handling of disagreement between annotators in the following sections.

5.3.2.1 Collecting Free News Articles

The goal is to create a dataset that can be openly distributed to foster future research. The general issue with news articles is that they are copyrighted. Thus, sharing the dataset easily is not possible. Spending a significant amount of time and effort to annotate data that cannot be used easily later on, would be a pity.

A typical workaround to this issue is to provide annotations independent of the text. Instead of the actual text, URLs are provided that need to be scraped by everyone who wants to use the dataset. However, this method often fails as either the article is not available anymore or does not produce the exact same text as articles are updated over time (or simply due to layout changes *etc.*). While these downsides might be acceptable for cheap document-level annotations (e.g. classifying the type of article), it is prohibitive for expensive token-level annotations. Even small changes to the text can render most of the offset-based annotations unusable as they are wrong without any indication.

Consequently, we choose a freely available and distributable news source: Wikinews articles are licensed under Creative Commons.

5.3.2.2 Source Data

The data originates from news articles published on the German Wikinews website. We used the XML (eXtensible Markup Language) dump⁶ available through the Wikimedia Foundation. Our dataset is based on the dump from April 2022 that consists of 13,001 published articles. From these published articles, we randomly sampled 1000 articles for annotation to stay close to the original distribution while reducing the data size to an amount manageable in our project timeframe. These articles range from December 2004 to March 2022.

5.3.2.3 Data Pre-Processing

To annotate on a token-level, plain text without errors are required. HTML-to-text approaches typically either result in somewhat messy output or missing pieces of content (or both). Wikinews also provides access to the raw text in MediaWiki’s markup language. As articles stored in MediaWiki markup contain custom macros for the German Wikinews, existing parsers do not work well. Thus, we developed a custom MediaWiki parser with support for the custom macros of German WikiNews. This was a rather tedious process as markup is often nested. Consequently, the conversion is a recursive procedure to support these nested macros present in the markup. Using this approach, we stripped all markup like formatting (*e.g.* bold, italic), semantic information (*e.g.* links to entities on Wikipedia) and non-textual content (*e.g.* pictures, tables) from the documents. Further, we removed any text not belonging to the main text body such as publication metadata, comments, links to related articles or sources. Altogether, it produces plain and simple text without errors or any unwanted components ready for our annotation project. The resulting plain text was tokenized and split into sentences using spaCy (Honnibal et al., 2020). The tokenized text was exported in the Webanno TSV format compatible with our annotation software.

5.3.2.4 Selecting the right Annotation Software

As the annotation schema is rather complex and challenging to apply efficiently, we spent some time evaluating different annotations tools to find the best-suited for our task. While we had to rule out many candidates due to licensing (*e.g.* Prodigy⁷), lack of fine-grained annotation spans or simply not being web-based so all of our annotators could access it (*e.g.* ATHEN⁸), we evaluated and compared the two most promising candidates in-depth with a small pilot annotation: INCEpTION (see Section 3.1.1, page 32) and CATMA (see Section 3.1.2, page 34).

CATMA does not support relations between different spans. While we considered enumerating the spans to group them together in post-processing, the annotators had a hard time seeing at a glance which span would be connected with another span.

INCEpTION does not support multipart spans. We worked around this limitation by connecting the individual components of multipart spans with relations. However, we also found that too many relations make the text almost unreadable slowing the annotators down significantly.

Overall, both candidates were no perfect fit for our annotation task. We chose INCEpTION with workarounds and some post-processing tricks to reduce the issue of too many relations hindering our annotators. This was far preferred by our annotators compared to CATMA

⁶URL: <https://dumps.wikimedia.org/dewikinews/>

⁷<https://prodi.gy>

⁸<https://gitlab2.informatik.uni-wuerzburg.de/kallimachos/Athen>

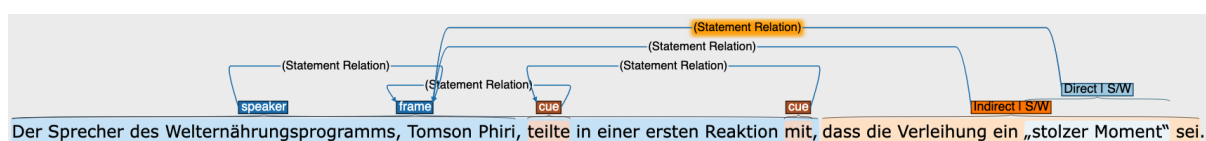


Figure 5.1: Excerpt from a document annotated with the full relations between all spans.

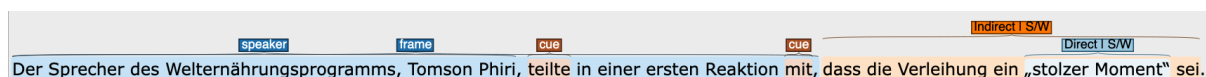


Figure 5.2: Excerpt from a document annotated in the simplified way that can be deterministically converted into the above full version by our custom post-processing software.

with enumerating spans to mimic some relation-like feature. In INCEpTION, we allowed our annotators to simplify annotations in unambiguous cases: Sentences with a single *Speaker* without relations to another sentence. We normalized these simplified, unambiguous sentences with a custom post-processing software to contain the same information as a sentence with full set of relations.

5.3.2.5 Annotation Process

Although we already expected the task to be too difficult for crowdsourcing⁹, it became apparent while evaluating the right tool for the annotation that crowdsourcing is no good fit for the annotation task with the goal to produce coherent, high-quality annotations due to the task’s complexity. Consequently, the annotation was carried out by three expert annotators with a background in German studies or Linguistics. The annotators were selected after performing a trial annotation on a handful of articles. The annotation team received extensive training during a preliminary annotation before the actual annotation begun. Further, we held weekly meetings during the main annotation to discuss open questions and uncertain cases as well as decide on the best solution, thereby providing both ongoing training to all annotators and improving the annotation guidelines as needed (e.g. for phenomena encountered only in the main annotation phase). The annotation quality of our annotators did not differ in a noticeable way after training. Neither in the discussions nor in the curation did it become evident that the annotations of one annotator were preferred over annotations of another annotator.

In an initial preliminary annotation, we iteratively tested which attributes of the Redewiedergabe schema are necessary and which additional options we needed. Finally, we settled on the medium and type attribute for a *Quote* and *Frame*, *Cue*, *Speaker* and *Addressee* as the other annotation components (roles). Details on the annotation classes are described in Section 5.3.1.

In INCEpTION, the different components are modeled as span annotations with relations between them to indicate e.g. which *Speaker* belongs to which *Quote*. We divided our sampled documents into six parts to

- a) annotate and curate in parallel,
- b) allow to adapt the annotation schema early in the process if needed, and

⁹The Redewiedergabe annotation project was performed by trained expert annotators

Part	Type	Medium	Roles
Part 1	0.56	0.37	0.61
Part 2	0.76	0.51	0.75
Part 3	0.77	0.40	0.76
Part 4	0.77	0.68	0.76
Part 5	0.86	0.51	0.83
Part 6	0.78	0.61	0.78

Table 5.3: Krippendorff’s Alpha agreement between the annotators on the six parts into which we split our whole corpus to work in parallel, adapt the schema if needed, and track inter-annotator agreement over time.

c) track the inter-annotator agreement over time.

We decided against automatic highlighting of candidate annotations *etc.* to not introduce any automatic processing bias. Thus, the instances were always manually identified by searching for the suitable grammatical structures.

5.3.2.6 Inter-Annotator Agreement

We use Krippendorff’s Alpha to compute the agreement between two annotators per part. The measure includes both the quality of the span annotation offsets (overlap) and their labels, but does not include the relations between the span annotations. However, the relations were typically made identical given the same annotation spans and their labels. Moreover, for different annotation spans, there is no sensible way to compute an inter-annotator agreement on the relations.

Table 5.3 shows the inter-annotator agreement values for the six parts into which we divided the 998 documents. The inter-annotator agreement values increased strongly after the first part, slightly increasing with additional experience and training over the course of the remaining parts. As such, the first part required significant curation effort and discussion that ultimately led to improved skills of our annotators. The inter-annotator agreement values for the medium fluctuate and show the lowest numbers in general because annotating the correct medium proved to be difficult depending on the context. The documents in part 3 (with the drop to 0.4) had many quotations where the medium was challenging for the annotators to select.

Neither the Redewiedergabe project nor Bögel and Gertz (2015) report inter-annotator agreement scores to compare to. As our annotation is a lot more complex than most span-based annotations (*e.g.* NER) it is to be expected that our scores are lower. With levels around 0.76 for type and roles we reached acceptable agreement; the scores are only slightly lower than the typical scores achieved in simpler span annotations tasks.

5.3.2.7 Disagreements between Annotators

During the annotation phase we held weekly meetings to discuss general questions how would we best annotate a specific phenomenon within our annotation schema. After two annotators had finished annotating the documents, we employed curation by a third person to resolve

Medium	Count	Avg. Len.
<i>Speech</i>	1906	16.75
<i>Writing</i>	572	19.13
<i>Thought</i>	2	10.5
<i>Speech/Thought</i> (ST)	322	14.95
<i>Speech/Writing</i> (SW)	1362	16.0
<i>Writing/Thought</i> (WT)	0	-

Table 5.4: *Quote* media statistics for our annotated dataset

differences in the annotations. In situations where the curator was not certain who (or if any) of the two annotators had correctly annotated the sentences in question, we discussed the issue in detail to resolve the disagreement, thereby potentially defining our annotation guidelines more precisely.

One of the most frequent reasons of disagreement during the early phases of the annotation was the difficulty of choosing the correct medium, usually the choice was between *Writing* or *Speech*. After many discussions, we concluded that it is sometimes impossible to decide from the text alone whether an utterance was produced in spoken or written form. As such, we modified our annotation schema by adding three new labels to medium. While this increased the annotation consistency considerably, it did not completely resolve the issue as the inter-annotator agreement shows.

5.3.2.8 Final Dataset

We exported and converted the curated articles into a JSON (JavaScript Object Notation) representation. During the conversion, we applied automatic checks for potential annotation errors and manually resolved true errors in the curated documents. The relations between the annotation spans allow us to build tuples where each tuple consists of one quotation with type and medium as well as all linked roles. Each text span is provided with character, token and sentence offsets enabling easy usage in various NLP frameworks.

5.3.3 Quantitative Analysis

In this section, we provide a quantitative view of the annotations in our dataset. Table 5.4 provides the total count and average length of *Quote* annotations for each medium. The analogous information for each quotation type is shown in Table 5.2. Table 5.1 provides the equivalent data for the role annotations. While most *Quotes* have a *Speaker*, only 70% have a *Cue* or *Frame*.

Figure 5.3 shows histograms of the token lengths for the different types of quotations. Overall, quotations lengths approach a heavily skewed normal distribution with some very short spans and a long tail for rare, long spans. Most quotations consist of 5 to 20 tokens. *Direct* quotations contain both the shortest and the longest quotations in the dataset. *Indirect* quotations are the shortest on average as they are usually fragments in a single sentence. *Reported*, *Free Indirect* and *Indirect/Free Indirect* (in increasing order of average length) have

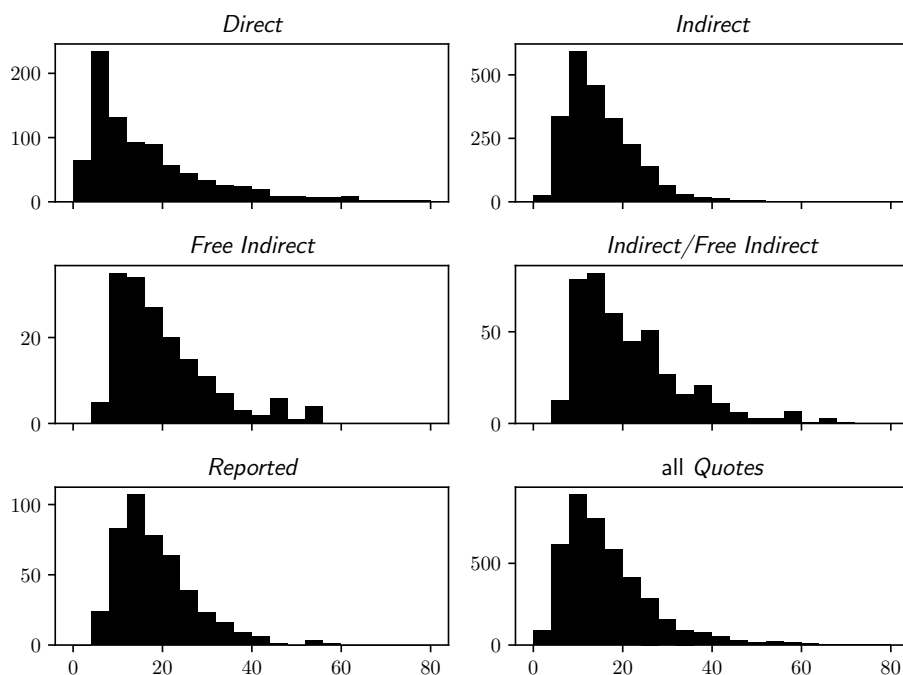


Figure 5.3: *Quote* token length histograms, taken from Petersen-Frey and Biemann (2024a).

similar distributions leaning towards longer spans since they normally consist of at least one but sometimes a few sentences.

Figure 5.4 shows the equivalent histograms for the roles. *Speaker* follows a Poisson distribution where most speaker spans are shorter than 5 tokens. Yet, some *Speaker* spans include descriptive phrases leading to more than 15 tokens (see Section 5.3.1.1 for details). The *Frame* annotations follow a skewed normal distribution like the *Quotes*. Since they can be a full sentence in length, they are the longest of the role spans with up to 40 tokens. *Cue* spans are the shortest annotations; typically a single token. However, around 33% of the *Cues* are multi-token expressions. The *Addressee* is a very rare annotation with lengths between one and eight tokens.

Figure 5.5 shows the number of quotations each role annotation is attributed to. In the overwhelming majority of cases each role span is only used for a single quotation. However, some spans are attributed to two *Quote* spans. This is especially true for the *Speaker* where up to five quotations are attributed to a single *Speaker* annotation.

We further analyzed the amount of nested *Quotes*, the number of sentences in a *Quote* and the distance between a role and its corresponding *Quote*. 10% of all *Quotes* are nested inside another *Quote* or *Frame*. The majority of these cases are instances of *Direct* speech fragments. While most *Quotes* span only one sentence, about 10% span two or more sentences (up to 11 sentences). 11% of all role spans and 21% of all *Speaker* spans are one or more sentences apart from the corresponding *Quote* (up to 7 sentences).

From these quantitative observations it becomes apparent that a system requires the following traits to be able to perform well:

1. Find *Quotes* and match the *Speaker* without any *Cue*.
2. Support for multi-word *Cue* spans.

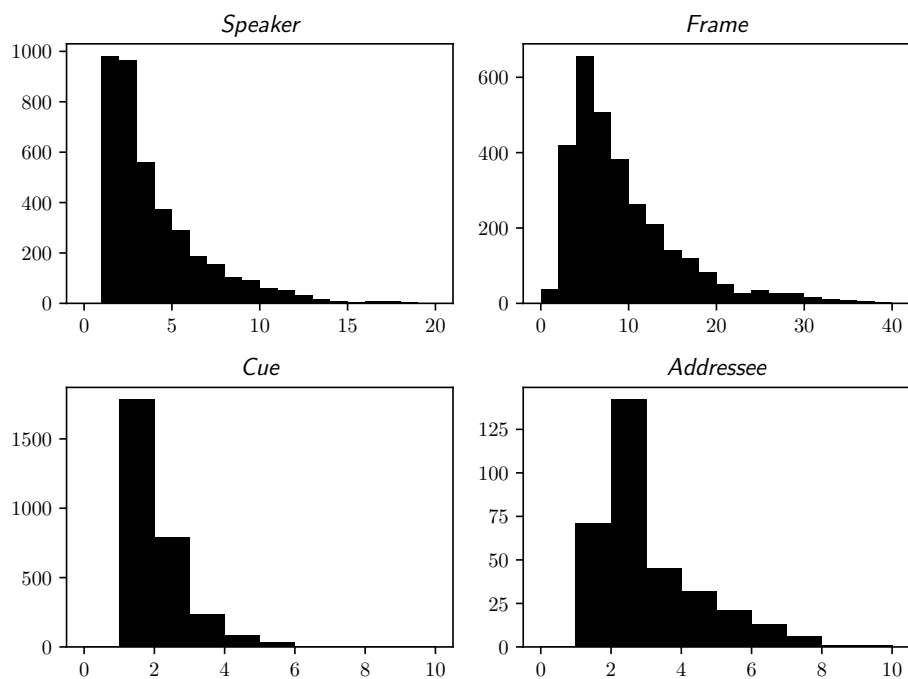


Figure 5.4: Role token length histograms, taken from Petersen-Frey and Biemann (2024a).

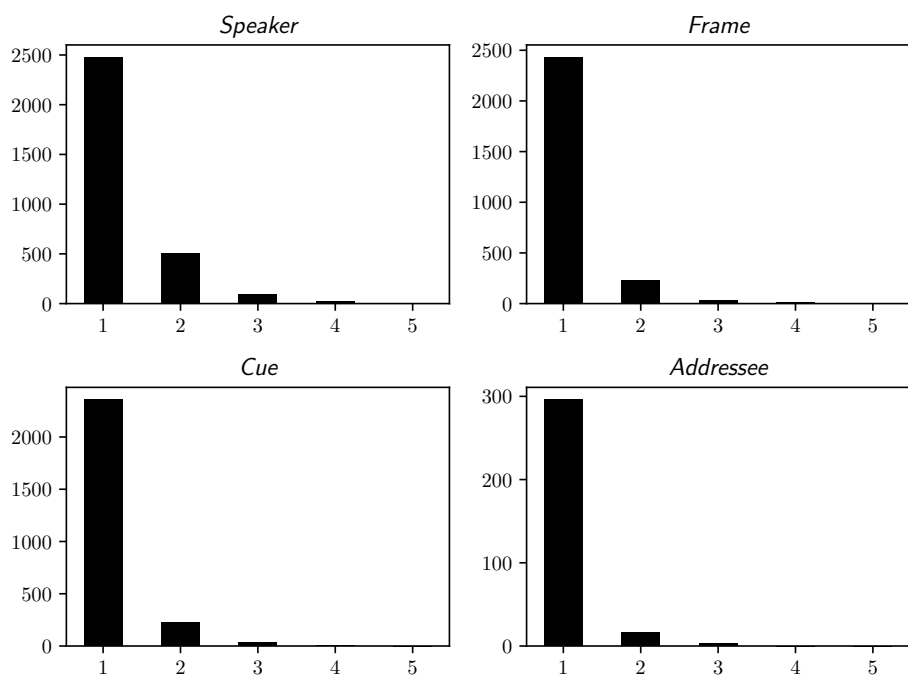


Figure 5.5: Number of Quotes per role span, taken from Petersen-Frey and Biemann (2024a).

3. Support *Quote* spans over multiple sentences.
4. Support finding roles for a *Quote* in other sentences.
5. Find *Indirect* and *Direct* quotations, also *Reported* and *Indirect/Free Indirect* as they account for 10% each, while *Free Indirect* makes up only 4% of all *Quotes*.
6. Support role spans to be used for multiple *Quotes*.
7. Support nested *Quote* spans.

5.4 Systems

An apparently similar problem to the task of quotation detection and attribution as defined in our dataset is SRL. Yet, we did not select SRL systems because the typical SRL datasets and thus available systems are limited to work on a single sentence as a unit. This is not suitable for the task that requires a document-wide context (or at least multi-sentence context) as quotes span multiple sentences and roles appear in yet other sentences.

In the remainder of this section, we introduce three systems that are able to perform quotation detection and attribution that we will apply on our dataset. We first introduce QUIiTE, our purely rule-based system that does not need any training data. Second we briefly describe the existing Citron (Newell et al., 2018) system that uses a data-driven ML approach. Finally, we extensively explain our novel Quotect system, a neural network (NN)-based model to perform fine-grained quotation detection and attribution.

5.4.1 QUIiTE: A rule-based System

As the only existing rule-based system by Bögel and Gertz (2015) is not available, we created a rule-based system to perform quotation detection and attribution: QUIiTE (QUotes in TExt). The advantage of a rule-based system is that it allows to quickly produce an automatic system that can detect some quotations and speakers without the need for any annotated training data. Further, it easily improves over time when better models for the individual components such as NER or dependency parsing become available. Another benefit is the ability to easily customize and tune a rule-based system according to the specific use case of our project partners in the CLICCS (Climate, Climate Change, and Society) B1 project¹⁰. As part of the project, news articles were to be analyzed to answer the overarching research questions which climate futures are possible and which are plausible according to news media.

5.4.1.1 Architecture

QUIiTE is build on top of spaCy (Honnibal et al., 2020) to extract both direct and some indirect quotations with the corresponding speaker from plain text. The system follows ideas of an older system presented by Bögel and Gertz (2015). It uses rules and word lists on top of neural components for dependency parsing, part of speech (PoS) tagging and NER. In contrast to Bögel and Gertz (2015), our system is open source¹¹, ready to use as a docker container¹², and uses neural NLP components in its pipeline for improved performance.

¹⁰<https://www.cliccs.uni-hamburg.de/research/theme-b/b1.html>

¹¹<https://github.com/fynn timeros/quite>

¹²<https://hub.docker.com/r/fynn timeros/quite>

Direct speech is identified by regular expressions looking for matching quotation marks. There are a number of quotations that use a specific character for the start or end of a quotation while other quotations are using the same character for both start and end of the quotation. The regular expressions take this into account and further consider the white space before or after quotation marks as well as any other punctuation like colons, commas, periods, quotation or exclamation marks. The *Speaker* of the quotation is searched in the proximity considering named entities and personal pronouns, preferring candidates in the same sentence but outside the quotation span.

Indirect speech is identified through the grammatical structure of a sentence (using dependency parsing) and the main or auxiliary verb being a *Cue* word that is looked up in a word list. The word list contains utterance verbs (*verba dicendi*) that can be used to indicate (in)direct speech. The sentence dependency parse is also used to find the subject and select it as the *Speaker* of the quotation. The *Indirect* speech span is build by following a relative clause of the *Cue* using the dependency parse.

In addition, the system finds sentences in subjunctive mood that occur directly before or after a sentence containing another quotation. These subjunctive mood constructions often occur in news articles. The *Speaker* of such a full-sentence quotation is taken from quotation of the previous or next sentence. These sentences are typically marked as *Indirect/Free Indirect* according to our later developed annotation schema. Lastly, the system combines overlapping *Direct* and *Indirect* speech, enriching the information of matching quotations.

5.4.1.2 Preliminary Evaluation

Most rule-based systems are inherently flawed: It quickly becomes impossible to match certain cases with rules but leave out others. In practice, tweaking one rule solves a specific prediction problem but tends to create a bunch of other performance degradations.

This issue became prevalent during our preliminary qualitative evaluation of QUITe during usage in the CLICCS B1 project. We applied QUITe on scraped news articles and manually gauged the quality of its predictions. For *Direct* quotations, it reached acceptable results for the quotation span. However, the *Speaker* was often wrongly predicted unless the *Speaker* information was transferred from an overlapping *Indirect* quotation. For *Indirect* quotations, both precision and recall are lacking for the quotation span. While some sentence constructions include a *Cue* verb from the list, they were not always a quotation leading to false positives. Vice versa, some quotations were not predicted because the sentence construction differed slightly from the implemented rules.

All in all, QUITe manages to predict straightforward cases well enough, but its prediction quality quickly diminishes for less common quotations. In general, the precision of the prediction seem to be better than the recall. This intuition matches the development context when implementing QUITe: It was build to extract quotations from large news articles collections without a need for high recall while also keeping the false positive rate low to limit the manual time spent on filtering false positives out. Ultimately, the quality of the predictions was too low to rely on QUITe. To quantify this preliminary, purely qualitative evaluation, we perform a thorough evaluation on gold-labeled data in Section 5.5.5.

5.4.1.3 Adapting QUiTE to our Dataset

We modified QUiTE to be evaluated on our newly created dataset. The system was not explicitly designed for the annotation schema of the dataset. Instead, we match the internal rules to the best-matching class in the dataset. Direct speech is supported by QUiTE, and we map this always to the *Direct* class. QUiTE has a number of rules producing indirect quotations. Normal indirect speech found using the word list of utterance verbs is mapped to *Indirect*. Sentences identified as indirect speech by QUiTE due to use of subjunctive mood are typically marked as *Indirect/Free Indirect* in the dataset; so we map the output to the *Indirect/Free Indirect* class. QUiTE does not handle the *Addressee* span. Since it is a rare class, we simply ignore it and do not predict any *Addressee*. However, *Frame* is a frequent role that the system predicts by marking all tokens of a sentence as the *Frame* that do not belong to the *Direct* or *Indirect* quotation. Since QUiTE does not predict a medium, we always use the majority class (*Speech*).

5.4.2 Citron

The Citron system was created by the BBC (Newell et al., 2018) to extract quotations with their *Speaker* from English news articles. It consists of several components that are built on top of spaCy (Honnibal et al., 2020) and are trained individually: *Cue* classifier, *Speaker* classifier, *Speaker* resolver, *Quote* classifier, *Quote* resolver. The resolvers link the classified spans to a *Cue*. The system can only find quotations that have a *Cue* — with the additional constraint that a *Cue* is single verb.

We modified Citron to work with German texts, accept any single token as a *Cue* and trained it using the subset of all quotations that have a *Cue* in our dataset (70%). As with QUiTE, the medium is always the majority class (*Speech*), we ignore any *Addressee* and predict the *Frame* to span all tokens in a sentence not belonging to the *Quote*. To predict the quotation type, we use *Direct* for spans with quotations marks and *Indirect* otherwise.

5.4.3 Quotect: Fine-grained Quotation Detection and Attribution

In this section describe our new ML based system to solve the task of fine-grained quotation detection and attribution with high quality. We developed a sequence-to-sequence transformer model with constrained decoding, and train it on our newly created dataset. Our model improves vastly upon our two baselines presented in the two previous sections, thereby making it feasible for a first time to reliably, extract attributed quotations automatically from German news articles. We provide an extensive description of our method by first providing an overview, explaining the forward transformation, specifying the constrained decoding strategy, and describing the reverse transformation. Then, we discuss alternative approaches, and present our ready-to-use Python library to easily perform quotation detection on plain text or pre-tokenized text in a handful lines of code. The training source code, inference package source code, and final trained models are publicly available.

5.4.3.1 Method Overview

We frame the task of detecting and attributing quotations as a special clustering task with labeled text spans. Each cluster contains a single *Quote* that may consist of multiple spans.

Further, a cluster may contain multiple spans of the different roles, *e.g. Cue, Frame, Speaker, Addressee*. In contrast to typical clustering, the same role span may belong to different clusters. We indicate this by applying multiple numeric identifiers to the same span.

On a high level, our method to solve the task of detecting and attributing quotations with a sequence-to-sequence model works as follows:

1. Transform the original text and cluster information to a suitable linearized token representation.
2. Train the sequence-to-sequence model with a language modeling objective to produce the linearized token representation from a plain text input.
3. During inference/generation, we constrain the model to produce a valid linearized token representation output.
4. We transform the linearized token representation back to the cluster information with token offsets in the original text.

In the next sections, we describe the relevant parts for this high-level scheme in more detail.

5.4.3.2 Linearization Strategies for Clustered Quotation Annotations

We transform this clustering into a sequential representation to employ standard sequence-to-sequence model to solve the task. Our method is inspired by Zhang et al. (2023) who show that it is feasible to perform state-of-the-art coreference resolution using standard sequence-to-sequence models with the appropriate textual representations. They experimented with a number of approaches to transform coreference mention spans and their clusters into linear text. With full linearization, the full input text is reproduced in the output — either as tokens or using a special copy action to prevent deviations between the input and target sequences.

With partial linearization, only the tokens being part of a mention span are reproduced in the output. This mode is incompatible with the copy action, potentially has alignment issues when matching the predicted output back to the original text and achieves slightly worse coreference resolution scores. Its advantage is a significantly shorter output length. As generating long token sequences is a slow and computationally expensive process with polynomial runtime, partial linearization provides a way to mitigate this issue. For coreference resolution, it requires only half of the time resp. compute compared to full linearization according to Zhang et al. (2023). The effect is more pronounced the fewer tokens are part of any annotated span, *i.e.* the sparser the annotations. While span-based coreference annotations often make up a substantial part of a document, the quotation annotations are often more sparse.

As such, we focus on the partial linearization outputting tokens with sentence markers for quotation detection and attribution to create a model usable with fewer computational resources. We call this method **partial token linearization**. Since full linearization with the copy action had the best results for coreference resolution, we use this as our second method for quotation detection and attribution named **full copy linearization**.

Zhang et al. (2023) also experimented with multiple techniques to include the cluster identifier for each span. For both our methods, we re-use their best-working policy and include the cluster identifier with a separation token | and a plain text integer number right before the closing tag of every span. Their integer-free representation alternative does not use integers in the text. Instead, it adds enumerated special tokens to mark a span as a mention and at the same time provide its cluster identifier.

Special token	Start	End
<i>Cue</i>	<cue>	</cue>
<i>Addressee</i>	<addr>	</addr>
<i>Speaker</i>	<speaker>	</speaker>
<i>Frame</i>	<frame>	</frame>
<i>Direct</i>	<direct>	</direct>
<i>Indirect</i>	<indirect>	</indirect>
<i>Reported</i>	<reported>	</reported>
<i>Free Indirect</i>	<frin>	</frin>
<i>Indirect/Free Indirect</i>	<infrin>	</infrin>
Copy from input	<cp>	
Cluster separation		
Sentence marker	<sent>	</sent>
Cluster identifiers	0	499

Table 5.5: Special tokens used

While integer-free representation is slightly shorter, it does not work as well with our linearization of labeled spans as it requires hundreds of special tokens for each type of span making it challenging for the model to learn so many new symbols with few training samples¹³.

5.4.3.3 Forward Transformation

Our transformation uses two special tokens per span type for both linearization methods (in addition to the separation and sentence marker tokens). We insert specific tokens that indicate the start resp. end of a span. Table 5.5 shows all special tokens used for our method. In total, we add 20 special tokens for full copy linearization, 21 special tokens for partial token linearization. For both, we also add 500 integer numbers for the cluster identifiers as special tokens to be represented by a single token in the vocabulary. To explain the forward transformation, we continue with Examples 5.1.1 and 5.1.2. We apply both linearization methods to produce a sequence out of this cluster information.

Example 5.4.1 shows the partial token linearization sequence. All tokens from the original text that are not part of any span are removed. For each sentence, a sentence marker is inserted at the beginning and end of it. Likewise, each span is marked using a special token directly before the text span begins. Spans are ended with a three-token sequence: Cluster identifier separation token |, cluster identifier, span closing token.

Example 5.4.1 (Partial token linearization):

<sent> </sent> <sent> <indirect> This is an indirect quote | 1 </indirect> <frame>,
 <cue> said | 1 </cue> <speaker> <speaker> someone | 1 </speaker> | 2 </speaker>

¹³The coreference resolution datasets for English (OntoNotes) and German (TüBa-DZ) consist of almost 2 million tokens each and boast a high ratio of mention spans per token while our quotation dataset has only 250,500 tokens with a low ratio of annotations per token

```
| 1 </frame> </sent> <sent> <direct> "Followed by a direct quote in a new sentence." | 2
</direct> </sent>
```

Again using the same inputs, Example 5.4.2 shows the full copy linearization sequence. Annotated spans are handled identical to the partial token linearization. In contrast, it does not contain sentence markers and all original tokens are still represented. However, they are all replaced with the special <cp> copy token.

Example 5.4.2 (Full copy linearization):

```
<cp> <cp> <cp> <cp> <cp> <cp> <cp> <cp> <indirect> <cp> <cp> <cp> <cp> <cp>
| 1 </indirect> <cp> <frame> <cp> <cue> <cp> | 1 </cue> <speaker> <speaker>
<cp> | 1 </speaker> | 2 </speaker> | 1 </frame> <cp> <direct>
<cp> <cp> <cp> <cp> <cp> <cp> <cp> <cp> <cp> <cp> <cp> | 2 </direct>
```

Although the resulting sequence markup reminds of XML, the partial token linearization does not result in valid XML due to overlapping tags, *e.g.* a <sent> may start within a certain quotation such as <direct> but end outside of it producing invalid XML such as <direct> . . . <sent> . . . </direct> . . . </sent>. For both linearization methods, nested spans are handled by first opening the span that ends later. As seen in Example 5.1.1, the *Frame* and *Cue* start at the same token. This results in a linearized sequence such as <frame> <cue> . . . </cue> . . . </frame>. The cluster identifier is always attributed to the span closed directly after the identifier. Together, this scheme enables a precise reconstruction of the original clustering information as long as opening and closing brackets match. It even works with nested spans of the same type, *e.g.* a *Frame* nested inside another, larger, *Frame* as long as opening and closing brackets match. To ensure this, we use constrained decoding.

5.4.3.4 Constrained Decoding Strategy

Sequences-to-sequence models can generate arbitrary output that might not be possible to convert back to a cluster representation. To prevent this and enforce valid outputs, we employ constrained decoding by masking the log-probabilities of vocabulary entries — a standard practice for sequence-to-sequence models (Daza and Frank, 2018; De Cao et al., 2021). We support two different modes (for partial resp. full linearization) that allow a varying degree of decoding constraints. We use beam search to improve generation quality. While it complicates the constrained decoding implementation, beam search has no effect on the rules we enforce; each beam will produce a valid output.

Partial token linearization Most of the time, we allow the model to generate any normal vocabulary sub-token, any special start-span token (so spans can be nested) or the | token used to separate span text from the cluster identifier. However, all these are forbidden after the | token has been generated before a special end-span token is issued. During this short time, only the cluster identifier and the correct end-span token can be generated by the model. The correct end-span token is the closing “bracket” for the most recently started span that is yet unclosed. The model may only generate the sentence start token after the token indicating the beginning of a document or end of a sentence.

Full copy linearization We never allow the model to generate a normal token. Instead, we only allow the special copy token to copy from the input or use the other special tokens to create spans resp. clusters. Similar to the first setting, we usually allow the model to generate the special copy token, any special start-span or cluster identifier separation token. The copy token is replaced with the actual token from the input when generating the next token. After the separation token only integers and the correct end-span token can be generated by the model until that span is closed. As we do not use sentence markers, no other special token except document end are allowed.

5.4.3.5 Reverse Transformation

Our system extracts a token offset-based cluster representation from the linearized sequence to identify the predicted spans and clusters in the original plain text.

Full copy linearization Since the constrained decoding as described above guarantees a valid output, we can transform the linearized sequence back to a cluster representation. As described in Section 5.4.3.3, nested spans are handled in a way so that it is always clear to which span a cluster identifier belongs and which span is ended (in case of identical span tags). To obtain the span begin and end offsets, we keep a stack of currently opened spans and close them in reverse order whenever a closing token is encountered in post-processing. Offsets are only affected by the number of copy tokens as only they correspond to sub-words in the original text; all other tokens are disregarded for this purpose.

Partial token linearization To find the offsets in the original text, the partial linearization requires an alignment by finding the sequence of predicted tokens of a span in the input text. We follow Zhang et al. (2023) who used Gotoh’s algorithm (Gotoh, 1982) together with sentence markers to efficiently find an optimal alignment. The sentence markers constrain the alignment to pairs of sentences. While the sentence markers are usually not required for a correct alignment of long *Quote* or *Frame* spans, the short role spans for *Cue*, *Speaker* and *Addressee* benefit as they often occur multiple times in the same document.

5.4.3.6 Alternative Methods Considered

To solve the task of quotation detection and attribution, we evaluated multiple approaches before we developed the system using a task-agnostic sequence-to-sequence model. In this section, we briefly describe what alternative approaches we considered and why we decided against implementing them.

Semantic role labeling At first sight, an obvious solution for the task is the re-use of an existing SRL system. However, they are strictly designed around the SRL task where roles must be identified for a single predicate (*Cue*) within a single sentence. This setup is quite incompatible with the task of general quotation detection and attribution where *Quotes* can occur without any *Cue* and often span multiple sentences. Consequently, we did not select with SRL systems to solve the task.

Coreference resolution with span tagging Another potential solution is to re-use a task-specific coreference model and combine it with a span tagging model. The coreference part can already handle nested spans and the clustering. However, a standard token tagging model is not enough to label the spans as this cannot support nested spans. The labeling part would

need to be deeply integrated in the coreference resolution model. Consequently, the system would no longer predict unlabeled mentions with arbitrary antecedent relations to form clusters. Instead, it would need to predict labeled spans with a specific set of allowed relations. While theoretically feasible, coreference resolution models include many special cases to cope with the computational complexity that make them ill fit for the quotation detection and attribution task. For example, word-level coreference (Kirstain et al., 2021; Dobrovolskii, 2021) is an efficient task-specific model that would be incompatible with the quotation detection task as the same head token can be used for multiple spans. Most architectures typically set a maximum span length to achieve practical computational complexity. This would also be an issue for the quotation attribution as the quotation spans vary greatly in length and can be much longer than a mention. Finally, quotations can span multiple sentences which does not occur with mentions and most systems rely on this fact.

Creating a new task-specific model architecture Adapting a coreference resolution model to properly fit the quotation detection and attribution task is arguably the creation of a new task-specific model architecture. We also decided against creating a new task-specific model architecture for two reasons. First, this approach makes the system dataset-specific to a large degree; thereby making the re-use of our model for similar datasets (e.g. in other languages) much more challenging. Second, the approach of creating task-specific model architectures for every NLP task is cumbersome, time-consuming compared to a modern, more generic solution.

5.4.3.7 Quotect Python Package

We make Quotect available as an easy-to-use Python package on PyPI¹⁴. In any Python environment it can be quickly installed with a straightforward `pip install quotect`. With just four statements, any text or pre-tokenized input is processed to perform quotation detection and attribution:

```
from quotect import Quotect, QuoteInput
qt = Quotect("fynnos/quotect-mt5-base")
input = QuoteInput(
    name="example",
    text="Sie sagte gestern: 'Morgen scheint die Sonne!' Jemand
        anderes erwiderte, dass es bestimmt regnen wird.",
)
print(qt.predict([input]))
```

The above code generates the following annotations visualized in JSON format (excerpt from the full output that also includes all individual tokens and sentences):

```
[
  {
    "speaker": {
      "spans": [{"charBegin": 0, "charEnd": 3, "begin": 0, "end": 1}],
      "text": "Sie",
      "tokenIds": [0]
    },
  ],
```

¹⁴<https://pypi.org/project/quotect/>

```

"frame": {
  "spans": [{"charBegin": 0, "charEnd": 18, "begin": 0, "end": 4}],
  "text": "Sie sagte gestern:",
  "tokenIds": [0, 1, 2, 3]
},
"cue": {
  "spans": [{"charBegin": 4, "charEnd": 9, "begin": 1, "end": 2}],
  "text": "sagte",
  "tokenIds": [1]
},
"quote": {
  "spans": [{"charBegin": 19, "charEnd": 46, "begin": 4, "end": 11}],
  "text": "'Morgen scheint die Sonne!'",
  "tokenIds": [4, 5, 6, 7, 8, 9, 10]
},
"type": "Direct"
},
{
  "speaker": {
    "spans": [{"charBegin": 47, "charEnd": 61, "begin": 11, "end": 13}],
    "text": "Jemand anderes",
    "tokenIds": [11, 12]
  },
  "frame": {
    "spans": [{"charBegin": 47, "charEnd": 72, "begin": 11, "end": 15}],
    "text": "Jemand anderes erwiderte,",
    "tokenIds": [11, 12, 13, 14]
  },
  "cue": {
    "spans": [{"charBegin": 62, "charEnd": 71, "begin": 13, "end": 14}],
    "text": "erwiderte",
    "tokenIds": [13]
  },
  "quote": {
    "spans": [{"charBegin": 73, "charEnd": 101, "begin": 15, "end": 20}],
    "text": "dass es bestimmt regnen wird",
    "tokenIds": [15, 16, 17, 18, 19]
  },
  "type": "Indirect"
}
]

```

With such an easy-to-use yet capable library users can either perform quotation detection and attribution in efficient batch processing by feeding in multiple QuoteInput objects at a time, use it interactively or integrate it into other software as a processing component. We see ease of use as a crucial feature to aid DH or CSS practitioners in applying ML models in their research. In Chapter 9 we show how Quotect is fully integrated into an open-source web application allowing to automatically process documents to detect and attribute quotations without any programming.

Split	Documents	<i>Quote</i> Count	Role Count
training	700	2877	7040
dev	150	653	1567
dev cue	150	445	1340
dev 1 cue	150	255	768
test	148	652	1605
test cue	148	468	1400
test 1 cue	148	300	899

Table 5.6: Dataset splits. We aimed to create two splits with a similar number of quotations and roles. The subsets “dev cue” and “test cue” contain only those quotations and their roles if a *Cue* is present. The subsets “dev 1 cue” and “test 1 cue” contain only those quotations and their roles if a single-word *Cue* is used.

5.5 Experiments

In this section, we describe the experiment setup and results on our dataset. We define suitable evaluation metrics, specify the dataset splits, list Quotect model variants to be tested, and describe training details for all three systems. Then, we discuss our main results, followed by ablation studies for all three models. Last, we evaluate the runtime efficiency of our Quotect model variants, and finally perform a manual error analysis of our models’ predictions.

5.5.1 Evaluation Metrics

We use the usual precision, recall and F1-metrics on token overlap of possibly discontinuous spans (thereby creating ordered sets of tokens). For most *Quote* types, all roles are optional. Thus, predicted spans of roles can only be matched to the reference roles if they belong to a correctly matched *Quote*. A span representing a role can be related to multiple *Quote* spans, *i.e.* the same *Speaker* can utter multiple *Quotes*. Roles or *Quote* spans can be nested within another *Quote* or *Frame*. Refer to Section 5.3.3 for statistics on these cases. To perform an evaluation, *Quotes* from system and reference are assigned via linear sum assignment of the *Quote* span’s token overlap using type and medium as tie-breakers. Each *Quote* can only be matched to at most one other *Quote*. The tie-breakers are needed to correctly assign *Quotes* in rare cases as they can have the same offsets, yet are of a different type or medium. If a system predicts a *Quote* that has no matching *Quote* in the reference annotations, this increases the false positives for *Quote* and each role the system predicted as belonging to the unmatched *Quote*. Vice versa, if a *Quote* from the reference annotation has no match in the system prediction, the false negatives are increased. A correctly matched *Quote* yields true positives for all correct roles according to the fraction of overlap and false negatives resp. false positives for tokens that were not identified resp. wrongly predicted by the system. Precision, recall and F1 are provided both independently for the quotations and roles as well as a joint metric.

5.5.2 Data

We split our dataset described in Section 5.3 into a training, development and test set for our experiments. The full dataset consists of 998 news articles. In total, these documents contain almost 250,000 tokens and 4,182 annotated groups. We aim for a 70%-15%-15% split regarding documents and number of groups. Finally, we split the documents via multiple rounds of random sampling into 700 training, 150 development and 148 test documents. Table 5.6 shows the basic statistics for the evaluation splits used in our experiments.

The data is available as JSON with the tokenized text and grouped annotations specified with token offsets for the span start and end. Each cluster always has a *Quote*; all roles are optional. For QUI TE, we only extract the tokenized text of the development and test sets as input for our experiments. For Citron, we write custom data loader classes to make Citron able to read the data format. As our Quotect model requires some pre-processing for training, we transformed the training dataset according to the forward transformation specified in Section 5.4.3.3. We transform the training split into different sequential representation for the training routine depending on the linearization method and the base model’s tokenizer.

5.5.3 Quotect Model Variants

We perform extensive experiments on a number of combinations of foundation language model, model size and linearization method. For the foundation language models, we test T5 (Raffel et al., 2020), T5 v1.1, Flan-T5 (Chung et al., 2022) as well as the multilingual mT5 (Xue et al., 2021). We test all models in three sizes: base (≈ 250 million parameters), large (≈ 800 million parameters), xl (≈ 3 billion parameters). While there are even larger models available, we do not have the computational resources to train such large models. Further, we test both of our linearization methods: Full copy and partial token linearization. In total, we test $4 \times 3 \times 2 \times 2 = 24$ combinations for our models. We could not obtain results on the two xl variants of the original T5 model because the computation ran into timeout errors on our available hardware. Thus, we only report results on the remaining 22 combinations.

5.5.4 Training Details

As QUI TE is a purely rule-based approach, it does not require any training. For Citron, we follow the training procedure of Newell et al. (2018): Using their default settings, we individually train *Cue* classifier, *Speaker* classifier, *Speaker* resolver, *Quote* classifier, *Quote* resolver. In the remainder of this section, we describe the Quotect training settings.

To enable efficient training in batches with low amount of padding and limited memory use, we use a sliding window approach and slice the training documents into chunks of 2048 subword tokens with an overlap of half the length. The evaluation and test documents are not sliced and are fed as whole into the system. Moreover, evaluation uses generation with 4 beams to improve prediction results.

We train each model for 100 epochs and use early stopping to prevent overfitting by evaluating on the development set. Weight decay of 0.01 is further used as a regularization method. For the base and large models we use a learning rate of $5 \cdot 10^{-4}$, for the xl models a reduced learning rate of $5 \cdot 10^{-5}$ as the models would not converge otherwise. AdamW

Model	Size	Lin.	Quotation			Roles			Joint			Type		
			Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
<i>development set</i>														
QUiTE			75.1	36.1	48.8	55.0	25.5	34.9	60.7	28.7	38.9	57.8	29.6	39.1
Citron			91.5	27.6	42.4	79.3	31.5	45.1	82.4	30.3	44.3	87.0	26.6	40.8
mt5	b	part	85.5	75.2	80.0	83.1	71.4	76.8	83.8	72.5	77.8	84.9	74.3	79.2
mt5	l	part	89.7	78.4	83.7	83.2	74.5	78.6	85.1	75.6	80.1	89.2	78.6	83.6
mt5	xl	part	86.4	79.5	82.8	82.5	75.8	79.0	83.7	76.9	80.1	85.3	79.0	82.0
<i>test set</i>														
QUiTE			70.8	36.2	47.9	55.6	26.1	35.5	59.9	29.0	39.1	63.5	33.6	43.9
Citron			88.2	30.1	44.9	77.9	34.2	47.6	80.5	33.0	46.8	86.5	29.6	44.1
mt5	b	part	85.9	77.2	81.3	81.8	73.2	77.3	83.0	74.4	78.4	87.5	78.1	82.5
mt5	l	part	88.8	81.1	84.8	83.0	75.2	78.9	84.7	76.9	80.6	89.5	81.0	85.0
mt5	xl	part	88.2	80.0	83.9	83.2	76.2	79.5	84.6	77.3	80.8	88.5	80.2	84.2

Table 5.7: Selected evaluation results (baselines and our best model combination per model size). Complete results are shown in Table 5.8 and Table 5.9.

(Loshchilov and Hutter, 2019) is used as the optimizer. To help the model adapt to its drastically changed task objective, we use a learning rate warm-up of 0.1.

We train with batch sizes of 8 using a single A100 GPU (Graphics Processing Unit) for base/large models. For the xl models, we use two A100 GPUs with Deepspeed (Rasley et al., 2020) ZeRO-2 (Rajbhandari et al., 2020) and a batch size of 4 per device, resulting in a total batch of 8. In all cases, we use gradient checkpointing and train the models in bfloat16.

5.5.5 Main Results

In this section, we present and discuss the results of our main experiments: Evaluating the three systems on our development and test splits. Table 5.7 shows the result of our best combinations per Quotect model size compared to two baselines for the development set (upper half) and test set (lower half) across all metrics except the medium. We do not report scores on the medium because no system is capable to predict it. The full results including all Quotect variants are shown in Tables 5.8 and 5.9. We discuss the different Quotect variants with the Figures 5.6 to 5.8 in our ablation study in Section 5.5.7.

From our results in Table 5.7 it is immediately obvious that our Quotect models far exceed the two baselines. Our best models deliver almost a two times higher F1 score than the baselines. While QUiTE and Citron obtain a joint F1 score of 39.1 resp. 46.8 on the test set, our best model scores 80.8. The difference in performance mainly originates from the vast increase in recall; going from 29.0 resp. 33.0 to 77.3 for our model. While we achieve a substantial improvement on the precision compared to QUiTE (60.7 versus 84.6), we manage a slight improvement over Citron (82.4 versus 84.6) — albeit at an entirely different recall level.

Model	Size	Lin.	Quotation			Roles			Joint			Type		
			Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
QUiTE			75.1	36.1	48.8	55.0	25.5	34.9	60.7	28.7	38.9	57.8	29.6	39.1
Citron			91.5	27.6	42.4	79.3	31.5	45.1	82.4	30.3	44.3	87.0	26.6	40.8
t5	b	full	82.4	68.3	74.7	76.2	61.9	68.3	78.0	63.8	70.2	81.6	68.0	74.2
t5	b	part	83.0	74.5	78.5	79.0	68.6	73.4	80.2	70.3	74.9	83.2	74.3	78.5
t5	l	full	86.7	70.0	77.5	80.1	63.2	70.7	82.1	65.2	72.7	87.6	70.4	78.1
t5	l	part	86.2	69.3	76.8	80.7	63.8	71.3	82.3	65.4	72.9	86.6	69.4	77.0
t5-v1-1	b	full	80.4	50.8	62.3	62.3	44.5	51.9	67.3	46.3	54.9	79.8	50.8	62.1
t5-v1-1	b	part	81.6	65.9	72.9	77.8	60.4	68.0	79.0	62.0	69.5	83.5	66.5	74.0
t5-v1-1	l	full	71.8	63.7	67.5	62.6	52.0	56.8	65.4	55.5	60.0	69.6	62.5	65.9
t5-v1-1	l	part	81.8	76.5	79.1	80.3	71.1	75.4	80.8	72.7	76.5	82.2	77.0	79.5
t5-v1-1	xl	full	85.9	74.0	79.5	80.6	71.3	75.6	82.1	72.1	76.8	86.5	75.3	80.5
t5-v1-1	xl	part	86.9	76.5	81.4	82.2	69.6	75.4	83.6	71.6	77.2	86.7	76.0	81.0
flan-t5	b	full	81.8	73.4	77.4	76.1	67.2	71.3	77.8	69.0	73.1	82.7	74.1	78.2
flan-t5	b	part	81.7	72.7	77.0	76.9	66.4	71.3	78.4	68.2	73.0	82.2	73.7	77.7
flan-t5	l	full	86.1	73.5	79.3	80.5	70.3	75.0	82.1	71.2	76.3	86.2	74.0	79.6
flan-t5	l	part	87.3	74.2	80.2	82.8	69.9	75.8	84.1	71.2	77.1	88.1	74.0	80.4
flan-t5	xl	full	89.0	77.3	82.7	83.9	73.3	78.2	85.4	74.5	79.6	89.6	77.6	83.2
flan-t5	xl	part	84.9	73.0	78.5	80.5	69.0	74.3	81.9	70.1	75.5	86.1	74.1	79.7
mt5	b	full	89.9	72.6	80.3	81.3	70.9	75.7	83.7	71.4	77.1	89.2	73.5	80.6
mt5	b	part	85.5	75.2	80.0	83.1	71.4	76.8	83.8	72.5	77.8	84.9	74.3	79.2
mt5	l	full	88.4	75.5	81.4	83.4	73.2	78.0	84.9	73.9	79.0	89.1	76.3	82.2
mt5	l	part	89.7	78.4	83.7	83.2	74.5	78.6	85.1	75.6	80.1	89.2	78.6	83.6
mt5	xl	full	84.8	78.8	81.7	82.2	76.5	79.3	83.0	77.2	80.0	84.5	80.2	82.3
mt5	xl	part	86.4	79.5	82.8	82.5	75.8	79.0	83.7	76.9	80.1	85.3	79.0	82.0

Table 5.8: Complete evaluation results on the development set. Highest score per metric marked in bold.

Regarding the prediction of the quotation type, our models again deliver almost twice the F1 score (44 vs 85). To put an F1 score of 80 into perspective: It corresponds to a nicely usable prediction from a manual evaluation of our model’s outputs, although there are small errors occasionally. We provide more impressions in our error analysis in Section 5.5.9.

When comparing the two baselines, the results show that both baselines achieve between decent and good precision while clearly suffering from low recall. Compared to Citron, QUiTE has lower precision, but higher recall of *Quote* resulting in a slightly better F1 score. For the roles, Citron has both better precision and better recall than QUiTE. Together (joint measure of *Quote* and roles), Citron again surpasses QUiTE in both precision and recall. As for predicting

Model	Size	Lin.	Quotation			Roles			Joint			Type		
			Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
QUiTE			70.8	36.2	47.9	55.6	26.1	35.5	59.9	29.0	39.1	63.5	33.6	43.9
Citron			88.2	30.1	44.9	77.9	34.2	47.6	80.5	33.0	46.8	86.5	29.6	44.1
t5	b	full	81.3	68.1	74.2	75.2	62.3	68.2	77.0	64.0	69.9	82.6	69.0	75.2
t5	b	part	84.7	72.9	78.4	79.6	66.4	72.4	81.1	68.2	74.1	84.5	72.7	78.2
t5	l	full	89.4	71.1	79.2	82.5	65.8	73.2	84.5	67.3	74.9	90.1	72.4	80.3
t5	l	part	86.2	72.2	78.6	80.5	65.6	72.3	82.2	67.5	74.1	86.3	72.4	78.7
t5-v1-1	b	full	77.8	53.7	63.5	61.6	46.3	52.9	66.0	48.4	55.9	80.7	55.1	65.5
t5-v1-1	b	part	83.8	67.2	74.6	78.4	61.9	69.2	80.0	63.4	70.7	85.2	66.9	74.9
t5-v1-1	l	full	71.4	62.8	66.8	63.3	54.3	58.5	65.7	56.8	60.9	71.2	63.7	67.2
t5-v1-1	l	part	83.5	77.1	80.2	78.0	70.0	73.8	79.6	72.1	75.7	84.3	77.3	80.6
t5-v1-1	xl	full	88.1	74.6	80.8	80.6	69.8	74.8	82.8	71.2	76.5	88.4	75.2	81.3
t5-v1-1	xl	part	89.3	76.8	82.6	83.2	70.5	76.3	85.0	72.3	78.1	89.6	77.0	82.8
flan-t5	b	full	83.0	73.2	77.8	73.3	66.0	69.4	76.1	68.1	71.8	83.9	74.2	78.8
flan-t5	b	part	81.0	75.5	78.2	74.9	67.2	70.8	76.7	69.6	73.0	82.4	76.2	79.2
flan-t5	l	full	86.6	76.8	81.4	78.9	73.2	75.9	81.1	74.3	77.5	86.8	77.5	81.8
flan-t5	l	part	89.4	77.9	83.3	82.5	74.0	78.1	84.5	75.1	79.6	90.1	78.2	83.7
flan-t5	xl	full	89.5	78.2	83.5	83.5	74.0	78.5	85.3	75.2	79.9	89.2	78.2	83.3
flan-t5	xl	part	89.9	76.3	82.5	84.2	70.2	76.6	85.8	71.9	78.3	90.7	76.7	83.1
mt5	b	full	91.2	75.4	82.6	82.6	71.4	76.6	85.0	72.5	78.3	90.0	74.5	81.5
mt5	b	part	85.9	77.2	81.3	81.8	73.2	77.3	83.0	74.4	78.4	87.5	78.1	82.5
mt5	l	full	89.5	78.3	83.5	84.1	74.4	78.9	85.6	75.5	80.3	90.3	78.7	84.1
mt5	l	part	88.8	81.1	84.8	83.0	75.2	78.9	84.7	76.9	80.6	89.5	81.0	85.0
mt5	xl	full	85.2	78.6	81.8	80.3	74.8	77.4	81.7	75.9	78.7	86.1	79.6	82.7
mt5	xl	part	88.2	80.0	83.9	83.2	76.2	79.5	84.6	77.3	80.8	88.5	80.2	84.2

Table 5.9: Complete evaluation results on the test set. Highest score per metric marked in bold.

the type of a *Quote*, QUiTE has slightly higher recall, but greatly lower precision than Citron leading to slightly better F1 score of Citron.

For QUiTE, the low recall mainly results from two causes. First, the system is not capable of predicting certain types of speech (*Reported* and *Free Indirect*) or roles (*Addressee*) that are present in the dataset. Second, the system was designed to prefer quality to quantity when automatically extracting quotations from large amounts of raw text. As such, the system has a preference for precision over recall even for types of speech that it can predict.

For Citron, the low recall has two reasons different reasons. First, the system only predicts quotes that have a *Cue* — but only 70% of all *Quotes* have a *Cue*. Second, the *Cue* recall itself is

System	Data	Quotation			Roles			Joint			Type		
		Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
QUiTE	dev	75.1	36.1	48.8	55.0	25.5	34.9	60.7	28.7	38.9	57.8	29.6	39.1
QUiTE	test	70.8	36.2	47.9	55.6	26.1	35.5	59.9	29.0	39.1	63.5	33.6	43.9
Citron	dev	91.5	27.6	42.4	79.3	31.5	45.1	82.4	30.3	44.3	87.0	26.6	40.8
Citron	test	88.2	30.1	44.9	77.9	34.2	47.6	80.5	33.0	46.8	86.5	29.6	44.1
Citron	dev cue	88.4	39.9	55.0	80.1	37.1	50.7	82.2	37.8	51.8	91.5	41.1	56.7
Citron	dev 1 cue	79.0	44.6	57.0	73.5	47.1	57.4	74.9	46.5	57.3	82.1	48.6	61.1
Citron	test cue	85.9	41.6	56.1	80.6	40.3	53.7	81.9	40.6	54.3	90.1	42.9	58.2
Citron	test 1 cue	80.0	47.6	59.7	74.7	48.3	58.8	76.0	48.3	59.0	85.4	50.7	63.6

Table 5.10: Ablation results on the modified dev and test splits

low because Citron’s *Cue* classifier a) cannot detect a multi-word *Cue* and b) was designed only for verbs as *Cue*.

Note that QUiTE is not tuned on the development set (and not even trained on the training set). Consequently, there should only be a small difference between the scores on the test and development set. However, there is a clear difference in the performance. This deviation in precision, recall and F1 between the test and dev dataset for QUiTE can be solely attributed to natural variations in the data, e.g. the different quotations contained in the documents of the test resp. dev set. The documents in the dev set contain quotations that happen to be more aligned with the rules implemented in QUiTE, thus reaching a slightly higher scores.

In summary, the data-driven ML-based Citron system approach outperforms the rule-based QUiTE system. Yet, our Quotect models far exceed the two baselines providing almost twice the F1 score. Both baselines suffer from low recall as they only detect a subset of all annotations. We analyze the effect from this in our dataset ablation in the next section.

5.5.6 QUiTE and Citron Ablation Study

To support our view and verify that the low recall of the Citron system is largely an effect of its *Cue* limitations, we performed additional experiments with modified versions of the dataset. First, we remove any quotations that do not have a *Cue*, i.e. that cannot be predicted by Citron. The results are shown in Table 5.10 (lower half) for the data *dev cue* (445 quotations, 1340 roles) and *test cue* (468 quotations, 1400 roles). Second, we further removed any quotations that have a multi-word *Cue* since Citron internally is limited to a single word *Cue*. We re-trained Citron with the new data and modified our QUiTE to only predict quotations with a *Cue*. The results are in the same table with the data column *dev 1 cue* (255 quotations, 768 roles) and *test 1 cue* (300 quotations, 899 roles).

For Citron, the effect is as expected: The recall for both quotes and roles significantly increases while precision slightly decreases, leading to increased F1 scores across the board. Limiting the dataset to quotations with any *Cue*, Citron sees +7.5 recall, +7.5 F1 on the dev set resp. +7.6 recall, +7.5 F1 test set joint scores combining quotations and roles. As a comparison,

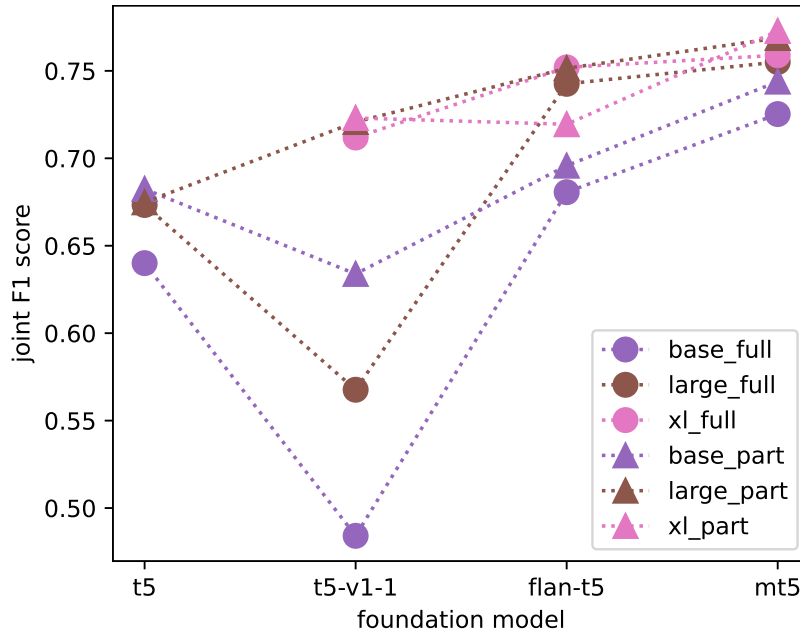


Figure 5.6: Joint F1 score per foundation model, taken from Petersen-Frey and Biemann (2024b).

for QUITe precision decreases (-9.3/-6.9), recall increases (+2.6/+2.2) and F1 remains unchanged (+0.0/+0.2) for dev/test joint scores. For limitation to a single-word *Cue*, QUITe’s performance degrades as its rule set is not compatible with the artificially reduced dataset: Joint precision -21.1/-19.3, recall +3.2/+1.8 and F1 -6.6/-6.0. Citron, however, improves another +8.7 recall, +5.5 F1 for the dev set resp. +7.7 recall, +4.7 F1 for the test set. Together, this results in an increase of +16.2 recall, +13.0 F1 on the dev set resp. +15.3 recall, +12.2 F1 on the test set joint scores. Thus, we attribute a large part of Citron’s low recall to its *Cue* limitations.

Most multi-word *Cue* expressions in the dataset are either past tense constructions or common idioms (see Section 5.3.1.1) that could be replaced by a single verb. When manually examining the data, there is no inherent difference in the difficulty between quotations and roles used in single- or multi-word *Cue* expressions. Consequently, it should be possible to achieve strong results on our whole dataset with a proper system; as evidenced by our well performing Quotect model.

5.5.7 Quotect Model Variants Ablation Study

In this section, we discuss the effect of the foundation model type, model size and linearization method. The Figures 5.6 to 5.8 compare the combinations only on the joint F1 score across the different dimensions. Figure 5.6 makes it easy to compare the performance across different foundation models. We can see a clear trend: mT5 outperforms Flan-T5 over all combinations of size and linearization method. Flan-T5 in turn outperforms T5 and T5 version 1.1 over all combinations. T5 version 1.1 behaves odd compared to the other three foundation models: It exhibits a rather large difference of over 20 F1 points between its best and worst combinations. We attribute this to the fact this specific foundation model was only pre-trained using language modeling excluding any supervised training on other tasks.

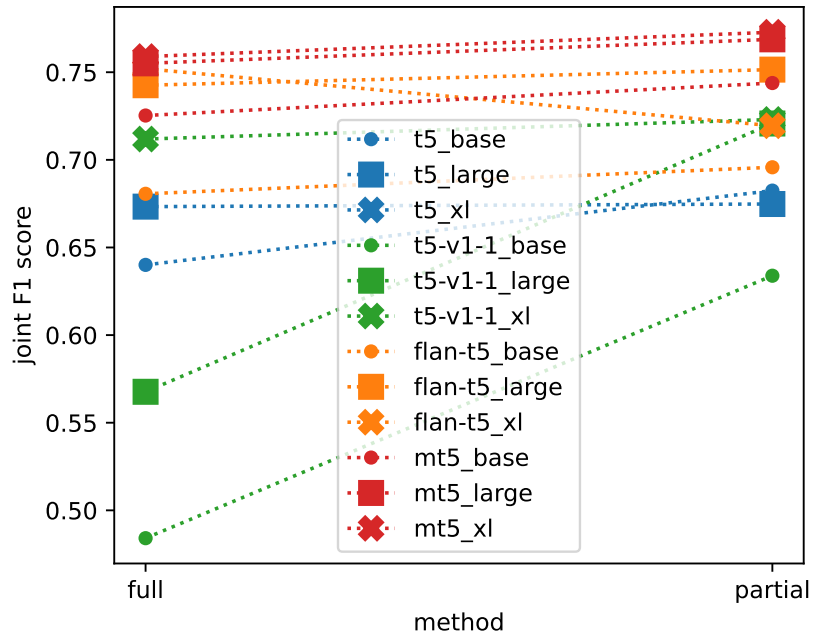


Figure 5.7: Joint F1 score per linearization method, taken from Petersen-Frey and Biemann (2024b).

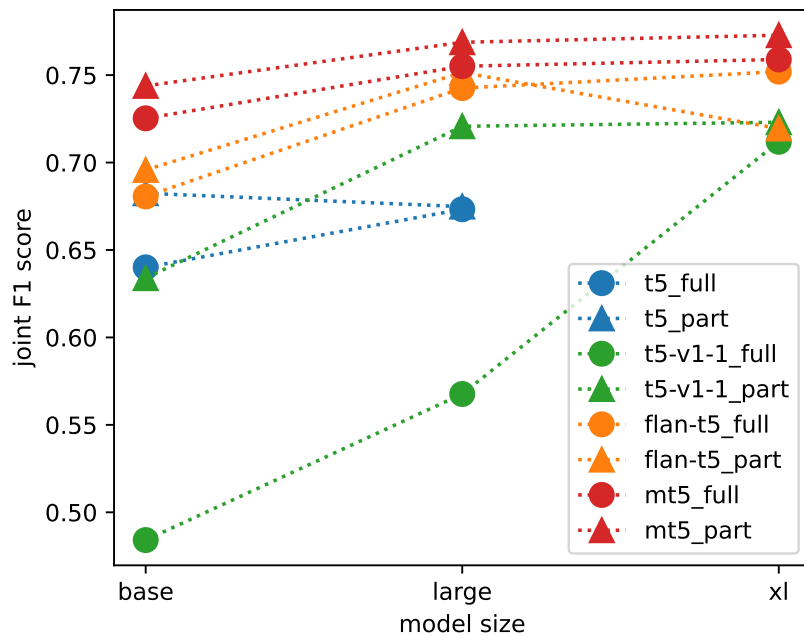


Figure 5.8: Joint F1 score per model size, taken from Petersen-Frey and Biemann (2024b).

Size	Linearization	Joint F1	Time	Time Batched
base	full	78.3	44m	24m
base	part	78.4	23m	13m
large	full	80.3	82m	60m
large	part	80.6	40m	31m
xl	full	78.7	90m	-
xl	part	80.8	49m	-

Table 5.11: Inference runtime

Figure 5.7 compares the performance across our two tested linearization methods: Full copy linearization and partial token linearization. In all but one cases, we see an increase in performance using the partial token linearization. The two smaller T5 version 1.1 models profit substantially from using the partial linearization. For most models, it provides a low, consistent performance boost — only for Flan-T5 XL the full copy linearization performs slightly better. Albert not directly comparable, this is in contrast to the results of Zhang et al. (2023) who reported a slight performance decrease in coreference resolution by using their partial linearization method. While the task of coreference resolution seems to be affected by the alignment issue inherent to any partial linearization, our task of quotation detection and attribution is less affected.

Figure 5.8 shows the effect of increased model size. While the large model variants noticeably outperform the base size equivalents for all but one combination, using the xl model instead of the large model only provides significant gains for the combination of T5 version 1.1 with full copy linearization. The Flan-T5 using the partial token linearization even shows a drop in performance. Consequently, we do not see a reason to use a xl model size as it requires significantly more computational resources than a base or large model.

5.5.8 Inference Runtime Efficiency

As we intend to use our system on large document collections, inference runtime and compute resource requirements play an important role. Consequently, we evaluated the runtime efficiency across the three model sizes and both linearization methods. Tests were performed using the mT5 models with batch sizes of 1 and 8 (only base and large). We ran the inference measurements on a single A100 GPU (80 GB) and predicted the development and test set together (298 documents). Table 5.11 shows the total time usage when predicting all documents including post-processing. Across all model and batch sizes, the partial token linearization is roughly twice as fast as the full copy linearization resp. needs only half of the computational resources. For the base model, batching is highly recommended as long as enough memory is available because it also reduces the inference time in half compared to no batching. The large model is 25% faster with batching. The base model is roughly twice as fast as the large model. Using a smaller model has the additional advantage of a lower memory usage allowing higher batch sizes with potentially even faster inference. The xl model is only 25% slower than the large model without batching because the xl model utilizes the GPU better.

When considering the quality of the predictions together with the required computational resources, only two options should be considered: Partial token linearization with either a base or large model depending on the preference for runtime performance or quality.

5.5.9 Error Analysis

We performed a manual verification of our best Quotect model’s output on both the development and test set. Beside the comparison with the gold annotations, we also checked whether false positives might be annotation errors. We found that the system sometimes has difficulties with the *Addressee* role: It often predicts the span as a *Speaker* instead. This is somewhat expected since *Addressee* is tiny minority class in the dataset. Similarly, *Free Indirect* is also a rare class and quotations of this type are either not predicted or used in debatable situations where the gold annotations did annotate the span, e.g. when the grammatical structure matches, but it represents the opinion of the article author.

It also occurred that multiple *Speaker* spans were predicted for a single *Quote* whereas the gold annotations only contain one *Speaker*. Further, the system sometimes predicts very short *Direct* quotes not annotated in the gold data. While some of these could be declared as annotation errors, most of these errors are justified. However, we also encountered multiple documents where the system predicted full quotations with roles that were most likely missed during the annotation. Another class of errors is that a role span is sometimes not used for multiple quotations, although the system correctly predicted all quotations — but without linking a *Speaker* etc. On a positive note, the system is perfectly capable of handling interrupted quotations and linking roles in a different sentence. Most nested quotations are also handled correctly, so the nesting itself does not appear to be a problem.

In general, it can be said that although the predictions are not perfect, they are reasonable and very much usable both for an in-depth analysis of extracted groups and a quantitative analysis over many documents.

5.6 Conclusion

In this chapter, we tackled the problem of automatically detecting who said what to whom in German news articles. To solve the task of quotation detection and attribution for German, we reviewed related work, created a new manually annotated dataset, created a rule-based approach by developing QUITe as a baseline, designed and trained our highly capable ML model Quotect, and finally performed and analyzed an extensive set of experiments of the models on our dataset showing excellent results of our Quotect model.

Our new dataset for quotation detection and attribution in German news articles is freely available under a Creative Commons license and provides curated, high-quality annotations.¹⁵ The fine-grained annotation schema allows the data to be used for various applications as it not only specifies who said what but also how, in which context, to whom and the type of quotation. We described our annotation schema and dataset creation in detail, performed a quantitative analysis, and evaluated two existing systems on our new dataset showing that a new approach is required to provide a high quality automatic detection of quotations.

¹⁵<https://github.com/uhh-lt/german-news-quotation-attribution-2024>

Quotect performs fine-grained, high-quality quotation detection and attribution in German news articles. Our system automatically identifies who said what to whom. The model predicts five different types of quotations together with the four different roles and connects these together. The model is based on a sequence-to-sequence transformer architecture generating structured, linearized output from plain text using constrained decoding. We described our method in detail, evaluated our two linearization methods across multiple foundation models and model sizes, performed an ablation study showing the importance of choosing the right foundation model, and performed a manual error analysis. Our models deliver a very strong performance on both a manual verification of the outputs and the evaluation metrics, almost doubling the scores of the other models.

Our Quotect system can be used for a range of use cases in the DH, CSSs and journalism. Especially when combined with additional NLP tasks such as coreference resolution and entity linking, identified quotations can easily be grouped and analyzed, thereby providing researchers and journalists new means to work with quotations in large document collections. To make the application easy for other researchers, we not only provide our training source code and trained models¹⁶ but also release Quotect as a ready-to-use Python package¹⁷ to perform quotation detection and attribution with only a handful lines of code.

5.6.1 Ethical Considerations

Automating tasks to scale to large data collections always carries a certain risk. ML models may have (unknown) biases due to their training data. In case of our Quotect model, the foundation language models have been pre-trained on a huge amount of diverse web texts (see Raffel et al. (2020), Chung et al. (2022), and Xue et al. (2021) for details).

Our task-specific fine-tuning is performed on the dataset that we created from a random sample of data of an open and freely available source without any focus on particular topics, speakers or authors that has been manually annotated and carefully curated — but certainly not error free. The articles in Wikinews might include certain biases as some articles will be written by the same authors, have the same source news agencies *etc.* Being an open, collaborative platform, the raw articles could still be less biased than relying only on licensed articles from one specific news outlet. The annotations in our dataset are likely neither perfectly error/bias free nor all-encompassing. Sometimes, it is a balancing act to decide whether a certain sentence contains a quotation or the article author only phrased the sentence in a certain way to suggest a quotation. Yet, we employed all means to create high-quality, fine-grained annotations to mitigate such issues by relying on skilled annotators, using annotation guidelines, weekly discussion meetings, curation and thorough handling of disagreements between annotators.

During a manual evaluation of our model's outputs, we did not encounter apparent biases such as a preference of gender for the speaker. Identifying who said what to whom according to news media on a large scale poses only a small risk compared to generating fake quotations with widely available large language models (LLMs). In any case, our dataset and trained Quotect model also provide the type of quotation so when using the identified quotations for a further analysis, it is possible to interpret the results more appropriately.

¹⁶<https://github.com/uhh-lt/seq2seq-quotation-attribution>

¹⁷<https://pypi.org/project/quotect/>

In summary, we do not see our dataset and model making things worse than the current state. On the contrary, our trained model can be used in helping to identify fake articles by comparing the found quotes with either a database or more reputable sources. Overall, the possibility to extract who said what to whom according to news media can be an invaluable tool for researchers and journalists in their work to analyze the vast number of online media, help with identification of fake news based on their quotations or ease verification of quotations.

5.6.2 Limitations

We see two limitations of our Quotect models. First, the computational resources required during inference are rather high as each generated token requires a pass through the decoder. We mitigate this by reducing the required output length with the partial token linearization. For documents that only include few quotations in the prediction, this makes inference very quick. For documents with many quotations, it is still faster than the full copy linearization although less pronounced. The second issue is the handling of rare events. Sometimes, deeply nested spans or span re-use across groups are not as well captured as they would be in isolation. Rare span classes such as *Addressee* have a lower recall compared to more common classes. More training data would help significantly with this issue. Due to the matching with the original input, our model remains free from hallucination in all situations.

5.6.3 The Case for Coreference Resolution

As shown in the experiments, quotation detection and attribution can be fully automated with both high precision and recall. In additional qualitative testing on news articles from other sources we found the model to still work very well. For practical application, one issue remains: Most *Speaker* spans are personal pronouns. This greatly reduces the utility of extracted quotations in follow-up analyses. Our models do not predict the actual *Speaker* name even if present in the text because the dataset annotations in most cases do not supply this information. Instead, the actual grammatical subject in the sentence (possibly the full entity name or just a pronoun) are annotated as *Speaker* for a good reason. Otherwise, the model would need to learn to implicitly perform coreference resolution with very few data points. Since that is prone to failure, we decided against this knowing that there are already large annotated datasets for German coreference resolution available — common datasets for coreference resolution are ten times larger than our quotation attribution dataset and contain many more annotations per sentence. Consequently, the final missing piece to proper fully automatic quotation detection and attribution is high-quality coreference resolution.

Combining our Quotect system with a well working coreference resolution system requires only minimal post-processing by matching *Speaker* spans with identified mention spans and thereby the entity. Compared to implicit coreference resolution in the quotation attribution model, the separation has a number of benefits: (1) models can be improved independently if better architectures are available, (2) error analysis is greatly improved, (3) the next step to entity linking or cross-document coreference resolution is already prepared. The downside is increased computation if the coreference information is not needed for anything else. As we plan to integrate both models into one common platform, the coreference information can very well be used for other analyses or automations. In the next chapter, we describe our approaches to state-of-the-art coreference resolution for German.

6 Transformer-based Coreference Resolution for German

Abstract Coreference resolution is the task of identifying and clustering text spans mentioning the same entities. The task is a widely applicable foundational part of a language processing pipeline to enable or improve other tasks. Previous coreference resolution approaches applicable to German are either impractically slow to compute or do not reach a quality required for fully automated processing. In this chapter, we successfully adapt two different coreference resolution architectures developed for English to German. Our new German coreference resolution models improve tremendously on the state of the art while being computationally efficient even for long documents. We perform experiments on two German datasets and ablations across different model variations, and analyze our models’ predictions in qualitative error analyses. Our source code and trained model weights are publicly available.

This chapter is (partly) based on the following publication(s):

Fynn Schröder, Hans Ole Hatzel, and Chris Biemann (2021). “Neural End-to-end Coreference Resolution for German in Different Domains”. In: *Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021)*. Ed. by Kilian Evang, Laura Kallmeyer, Rainer Osswald, Jakub Waszczuk, and Torsten Zesch. Düsseldorf, Germany: KONVENS 2021 Organizers, pp. 170–181. URL: <https://aclanthology.org/2021.konvens-1.15/>.

Fynn Petersen-Frey, Hans Ole Hatzel, and Chris Biemann (2025). “Efficient and Effective Coreference Resolution for German”. In: *Proceedings of the 21st Conference on Natural Language Processing (KONVENS 2025): Long and Short Papers*. Ed. by Christian Wartena and Ulrich Heid. Hannover, Germany: HsH Applied Academics, pp. 128–137. URL: <https://aclanthology.org/2025.konvens-1.12/>.

6.1	Introduction	106
6.2	Related Work	107
6.2.1	Coreference Resolution for English	107
6.2.2	Coreference Resolution for German	109
6.3	Datasets	109
6.4	Models	110
6.4.1	Coarse-to-Fine	111
6.4.2	Maverick	113
6.4.3	Model Architecture Comparison	115
6.5	Experiments	116
6.5.1	Evaluation	116
6.5.2	Preliminary Model Design Experiments: Coarse-to-Fine	117
6.5.3	Preliminary Model Design Experiments: Maverick	118
6.5.4	Main Experiments	119
6.5.5	Error Analysis	121

6.6	Conclusion	122
6.6.1	Ethical Considerations	123
6.6.2	Limitations	124

6.1 Introduction

Coreference occurs in a text when two or more phrases, called mentions, refer to the same entity, such as a person, object, or event. A mention refers back to a previous text span, called its antecedent, which references the same entity. Coreference resolution deals with resolving these references by connecting each mention with its antecedent. In more technical terms, it is the task of identifying and clustering mentions in the form of text spans in a document referring to an entity. All mentions referring to the same entity need to be assigned to the same cluster to build mention clusters where each cluster represents exactly one entity. Figure 6.1 shows a short example with resolved coreference annotations.

The task of coreference resolution is a widely applicable foundational part of a natural language processing (NLP) pipeline to enable or improve other tasks such as question answering (Morton, 1999), text summarization (Steinberger et al., 2007), quotation attribution (see Chapter 5, page 67), or aid in literary analysis (Bamman et al., 2014), discourse analysis, *etc.* While our main motivation is to greatly improve the usefulness of the automatic speaker attribution of quotes, a reliable coreference resolution model is helpful in many situations involving text processing or analysis.

Coreference resolution has been extensively researched in the past. While early models relied on rule-based approaches, current state-of-the-art systems are built as end-to-end neural models leveraging large pre-trained language models (PLMs). As a result, the quality of the system’s outputs has drastically improved over the last years. In the English CoNLL-2012 shared task (Pradhan et al., 2012), a rule-based system achieved a CoNLL-F1-score of 57.8. The first end-to-end neural system (Lee et al., 2017) has increased this score greatly to 67.2, the current state of the art (SotA) is 83.3; thereby showing that neural systems are key to state-of-the-art performance as in many other NLP tasks. However, the computational demands have also sharply increased, rendering many approaches impractical for use with large document collections, especially with longer documents, as runtimes typically increase superlinearly.

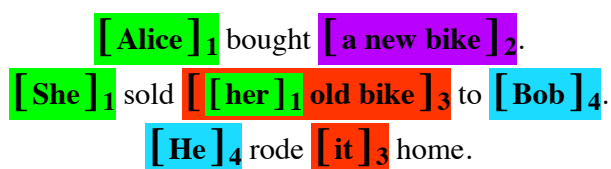


Figure 6.1: Visualization of coreference mentions and cluster information. Mentions are marked by indexed brackets and background color that both also denote which cluster the mention belongs to. *Alice*, *She* and *her* form the first cluster. *a new bike* is a singleton mention that belongs to cluster two without any other mention. *her old bike* and *it* form cluster three. *Bob* and *he* are the mentions in cluster four. Figure taken from Petersen-Frey et al. (2025).

The majority of the research has focused on English as a target language, with only few approaches targeting German.

In this chapter, we successfully adapt two existing approaches for English coreference resolution to German. We make the following contributions:

- First neural network (NN)-based German coreference resolution system.
- New state-of-the-art coreference resolution models for German news with tremendous quality improvements
- that are also computationally efficient with long texts.
- Ablation experiments across various model parameters and design choices.
- A thorough evaluation of our models by comparison with existing models and an error analysis.

In the following sections, we provide an overview of related work, describe relevant datasets, explain our two models in detail, describe our performed experiments and analyze their results, and finally conclude our findings.

6.2 Related Work

A wide body of work has dealt with the problem of coreference resolution. Rule-based approaches have long been used to address the task of coreference resolution (Raghunathan et al., 2010; Lee et al., 2011). However, relevant prior work can mainly be put into two distinct categories: (a) neural, state-of-the-art coreference resolution developed primarily on English, and (b) coreference resolution applied on German.

6.2.1 Coreference Resolution for English

Neural coreference resolution models are either encoder-based discriminative systems or decoder-based, generative large language models (LLMs). The majority of encoder-based models perform a ranking of antecedents based on the pairwise scores of mention candidates (Wiseman et al., 2015; Clark and Manning, 2016a; Lee et al., 2017), thereby only relying on local decisions that may not be globally optimal to form coherent entities (Lee et al., 2018). Advancements in neural architectures, specifically encoder models, have enabled end-to-end neural systems which generally follow the mention linking approaches but differ in architectural details and in how they represent mentions (Lee et al., 2017, 2018; Dobrovolskii, 2021; Kirstain et al., 2021). This general architecture has been improved on in multiple ways.

To address the issue of global optimization, Clark and Manning (2016b) and Wiseman et al. (2016) create entity representations during the ranking step. Lee et al. (2018) and Kantor and Globerson (2019) iteratively refine mention representations with associated antecedent information, performing what they refer to as higher-order inference.

While the end-to-end coreference model of Lee et al. (2017) uses a bidirectional long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) to produce span representations, Lee et al. (2018) see a 3.2 F1 score increase on the English CoNLL-2012 shared task by additionally using ELMo (Peters et al., 2018) embeddings. Lee et al. (2018) also modify the model to perform coarse-to-fine antecedent pruning enabling an efficient computation and potentially allowing the processing of longer documents. Joshi et al. (2019) and Kantor and Globerson (2019) improve upon this by using BERT (Bidirectional Encoder Representations from Transformers)

(Devlin et al., 2019) embeddings instead of the LSTM-based representations and gain another 3.3 F1 points.

Recently, Joshi et al. (2020) presented a model optimized for span representations named SpanBERT and saw another 2.5 point increase in F1 score, which has been reproduced by Xu and Choi (2020). Wu et al. (2020a) have taken a different approach to coreference resolution; they outperform previous state of the art by 3.5 F1 points in part due to the ability to recover missed mentions by framing the task as a question-answering problem. They also report a strong performance improvement using SpanBERT instead of BERT.

Toshniwal et al. (2020) and Xia et al. (2020) both introduce incremental approaches to coreference resolution. Instead of comparing mention pairs like Lee et al. (2017), they compare mentions with entity representations, with the entity representations being produced from a linear combination of their mentions. Both approaches work by iteratively processing all mentions and scoring each mention with regard to a set of entities; as a result, an evaluation of the full cross-product of mentions is not necessary. The two approaches differ slightly in how they handle the introduction of new entities. Xia et al. (2020) either add a new mention to the entity list (using the mention representation as the initial entity representation) or update one of the existing entity representations using a learned update gate. Toshniwal et al. (2020), by contrast, explicitly decide between either updating an existing entity, creating a new one, or discarding the mention thereby enabling them to handle singleton mentions, those mentions that are the only ones in the same document referring to a specific entity.

Dobrovolskii (2021) and Kirstain et al. (2021) introduce two different approaches to solve the computational issue of creating distinct mentions representations for all possible mention candidates and comparing them pairwise. Dobrovolskii (2021) circumvent the expensive $O(n^4)$ comparison of all mention candidate representations by resolving coreferences between individual words in $O(n^2)$ instead. Afterwards, they reconstruct the actual mention spans for each pair of linked words. The word-level approach cannot work with multiple mention spans having the same head word and therefore has issues with nested spans, according to D’Oosterlinck et al. (2023) who also propose a mitigation strategy for common cases by selecting the conjunction as head word. Kirstain et al. (2021) take a different approach to reduce the number of antecedent comparisons by representing mentions implicitly by their start and end token embedding. Instead of creating new explicit span representations, they resolve coreferences by summing four pairwise comparisons of token representations in $O(n^2)$: start-to-start, start-to-end, end-to-start, end-to-end of each candidate span.

In recent years, text-to-text approaches have increasingly been used to essentially perform inline coreference annotations (Wu et al., 2020a; Bohnet et al., 2023; Zhang et al., 2023). While such text-to-text approaches yield very strong results, recent advances in encoder-based approaches have caught up in terms of resolution quality while offering vastly quicker runtime performance (Martinelli et al., 2024). Recent work has called into question whether coreference resolution architectures have improved over time, attributing the gradual improvement mainly to the underlying language model’s sizes instead (Porada et al., 2024). The text-to-text approaches rely on much larger models than the encoder-based systems to achieve their resolution quality levels.

6.2.2 Coreference Resolution for German

For coreference resolution on German texts, specific rule-based and machine learning (ML) approaches have been developed. The CorZu system (Klenner and Tuggener, 2011; Tuggener and Klenner, 2014) is a rule-based incremental entity-mention model that has been extended with Markov Logic Networks (Taskar et al., 2002) for the antecedent selection. According to the authors, it was the first ready-to-use coreference resolution system for German. In the system, potential mentions are compared to existing coreference chains and other yet unused mentions to build a set of antecedent candidates. From this set of candidates the best antecedent is selected via a rule-based salience calculation or Markov Logic Networks. The mentions consist of nouns, pronouns and named entities that are extracted in a preprocessing step producing part of speech (PoS) tags and a dependency parse. Krug et al. (2015) devised a rule-based system explicitly targeting German historic novels. Their approach features 11 rule sets that are applied sequentially as every later rule set relies builds on the results of earlier rule sets.

IMS HotCoref DE (Roesiger and Kuhn, 2016) is adapted to German from the English system of Björkelund and Kuhn (2014), a tree-based ML approach using handcrafted features. A directed tree where each node represents a mention is used to model the coreferences in a document. For determining antecedents, both local and non-local handcrafted features are employed. During training, latent trees are induced via beam search to generate gold structures, as the training data only provides mentions and their corresponding entity. In order to be applied on raw text, the system requires an elaborate preprocessing pipeline adding PoS tags, lemmata, constituency parses, number and gender information as well as named entities to the input. They created the previous state-of-the-art approach for the German TüBa-D/Z dataset.

Apart from our own published works, two other works have also targeted German coreference resolution using NNs. Bohnet et al. (2023) apply a transition-based LLM coreference system on, among other languages, German. Their approach requires enormous computational resources — 8 TPUs (Tensor Processing Units)-v4 for inference, 128 TPUs-v4 for training, and uses additional training data as they continue training their German model from the pre-trained English coreference resolution model. Gupta et al. (2024) focus specifically on literary texts, contributing a novel technique for handling texts with lengths that exceed the underlying encoder model’s context.

6.3 Datasets

In terms of datasets, there are two German datasets that are relevant to our work: (1) TüBa-D/Z (Telljohann et al., 2004) and (2) SemEval 2010 (Recasens et al., 2010). The standard corpus for coreference resolution in German is TüBa-D/Z, a manually annotated collection of newspaper articles released in multiple versions that incrementally add more documents. It was also used as the data source for the German part of the SemEval-2010 shared task on coreference resolution (Recasens et al., 2010).

According to the coreference annotation guidelines (Naumann and Möller, 2006; Telljohann et al., 2017) mentions (definite noun phrases, personal pronouns, relative, reflexive, and reciprocal pronouns, demonstrative, indefinite and possessive pronouns) are extracted

Dataset	Articles	Sentences	Tokens
SemEval-2010	1,235	26,098	455,046
- training	900	19,233	331,614
- development	199	4,129	73,145
- test	136	2,736	50,287
TüBa-D/Z 10.0	3,644	95,595	1,787,801
- training	2190	65,416	1,258,514
- development	727	15,593	276,635
- test	727	14,586	252,652
TüBa-D/Z 11.0	3,816	104,787	1,959,474
OntoNotes 5.0 (English)	3,493	94,269	1,631,995

Table 6.1: Overview of the coreference datasets referred to in this chapter together, and their data sizes in number of articles, sentences and tokens.

automatically from the treebank information. Noun phrases are annotated with the maximal extension including complements, adjuncts and any contained statements in brackets.

Table 6.1 shows basic corpus statistics for the datasets mentioned in this work. To be comparable with previous work, we chose to use SemEval-2010 and TüBa-D/Z release 10.0 instead of the marginally larger 11.0 for our experiments. As there is no official split for the TüBa-D/Z, we use the same splits as previous work (Roesiger and Kuhn, 2016) and thus use the first 727 documents for testing, the next 727 for development and the remaining 2190 documents for training.

While TüBa-D/Z does not contain singletons (on average 3.65 mentions per entity, 10.89 entities per article), singleton mentions are annotated in SemEval-2010 (on average 1.34 mentions per entity, 73.07 entities per article). Across the dataset, 84.6% of all entities and 64.1% of all mentions are singletons.

Compared to the standard English coreference corpus, OntoNotes (Weischedel et al., 2013), used in the CoNLL-2012 shared task on coreference resolution (Pradhan et al., 2012), TüBa-D/Z neither contains different genres of texts nor additional metadata such as speaker information. Regarding statistics such as average mentions per entity, mentions/sentence length and tokens/sentences/entities per document, German TüBa-D/Z 10.0 and English OntoNotes 5.0 are remarkably similar.

6.4 Models

In this section, we describe two models that we developed to perform coreference resolution on German news texts. Both models rely on transformer-based encoders and their architecture is adapted from related work originally developed for English. They perform mention detection and antecedent matching differently, and feature different strengths and weaknesses. We provide a comparison of both models in Section 6.4.3.

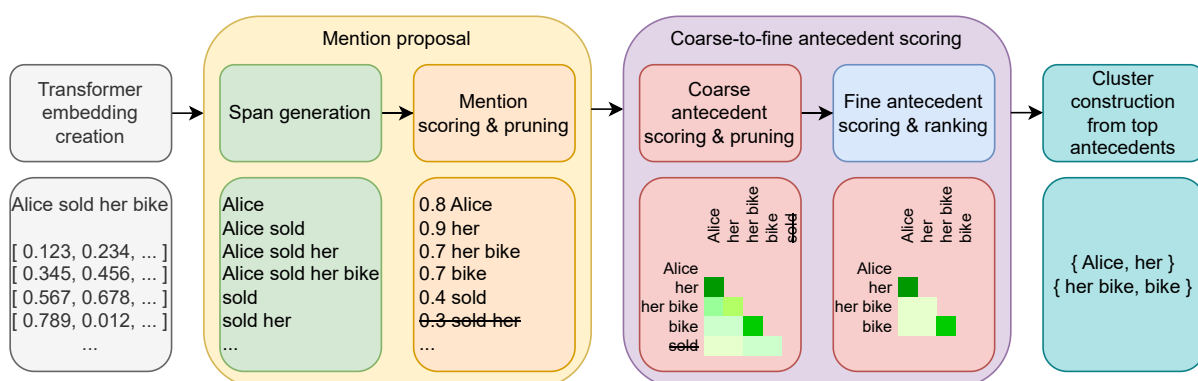


Figure 6.2: Conceptual visualization of our coarse-to-fine model architecture processing a tiny example document (“Alice sold her bike”). The model generates possible spans for mentions candidates, scores and prunes them. Then, it performs one comparison on the cross product of all mentions before the fine antecedent scoring and ranking. Dark green colored squares indicates a good match between mention and its assignment candidate, whereas brighter colors represent a lower score. All values are manually chosen for illustration purposes. Figure inspired by Schröder et al. (2021).

6.4.1 Coarse-to-Fine

Our first model is a coarse-to-fine model described in detail in this section. We build on the widely adapted neural end-to-end architecture developed by Lee et al. (2017, 2018), improved by Joshi et al. (2019) and re-implemented in PyTorch (Paszke et al., 2019) by Xu and Choi (2020). Although the CorefQA system (Wu et al., 2020a) is currently the top-performing system for English, we chose to not build upon it because it is more complex and requires vastly more computational resources (they list a TPU with 128 GB memory as a requirement for training¹) than our chosen approach.

The overall idea of the model is to first detect mentions candidates and then link them together thereby forming mentions clusters corresponding to individual entities. Each document is processed individually during both training and inference. Figure 6.2 shows how the model processes an individual document. Initially, contextual ELECTRA (Clark et al., 2020) embeddings are obtained for each token and all possible mention spans up to a configurable maximum length $M = \text{max_mention_length}$ are generated. This maximum length of mention spans serves two purposes: First, it reduces the computational complexity by limiting the amount of possible mention spans. Second, it provides an inductive bias for the training because mention spans are typically shorter than a sentence. Long documents are split into fixed-size segments and embeddings are computed independently for each segment as suggested by Joshi et al. (2019). This helps to keep the expensive transformer attention mechanism within reasonable bounds. Then, mention embeddings are created that are the concatenation of start and end token embeddings together with the attention-weighted average of all span tokens. In contrast to the English models, our models contain neither genre nor speaker embeddings as the German datasets do not supply this information.

¹See headline “Hardware Requirements” at <https://github.com/ShannonAI/CorefQA>

A naïve approach of performing a pairwise comparison of all mention candidates to find links between them causes computational issues. It quickly becomes infeasible to compute this pairwise similarity as it requires $O(M^2)$ comparisons for $M = \text{max_mention_length} \cdot |D|$ mention candidates, for a document D where $|D|$ is the document length in word-piece tokens. To reduce the computational requirements over a naïve approach to find the best antecedent for each mention, we employ a coarse-to-fine pruning approach.

Our model implementation is based on the implementation by Xu and Choi (2020). For each mention span, the model learns a distribution over its antecedents based on how likely both individual spans are to be valid mentions and how likely they to refer to the same entity. Two pruning steps are used to make this mention linking computationally feasible.

To reduce the number of mentions, all mention embeddings are scored individually with a feedforward neural network (FFNN). For each document D only the top $n = \min(4096, 0.4 \cdot |D|)$ mentions are kept after pruning. Instead of performing a pairwise comparison of all N mentions, only a fraction is used. Thus, removing obvious non-mentions, preventing a full cross-product comparison, and thereby limiting the number of comparisons to $O(n^2 \ll N^2)$, a step that we refer to as mention filtering.

In the coarse antecedent pruning step, the pairwise similarity scores of the remaining mention embeddings are summed with the individual mention scores. A subsequent fine-grained ranking is performed with the top $a = 64$ antecedents per mention; to this effect, pairwise mention-antecedent embeddings consisting of mention, antecedent and similarity embedding are created. These embeddings are scored with a FFNN and combined with scores from the coarse pruning step resulting in scores for the top antecedents per mention. We do not use so-called higher-order inference as this effectively doubles the computational cost of the fine-grained antecedent scoring without improving the quality according to Xu and Choi (2020).

During training, the model learns to optimize the marginal log-likelihood of possibly correct antecedents for each mention, *i.e.* for each antecedent the score should be 1 if mention and antecedent belong to the same gold entity, 0 otherwise. This marginalization is required because the best antecedent is a latent variable. Although the model consists of a multistep process, we train and predict it in an end-to-end fashion, *i.e.* the mention detection step is trained jointly with the antecedent matching step. It is possible to first individually pre-train the mention detection step of the model — however, this requires all mentions to be annotated in the training dataset (which is not the case for the TüBa-D/Z dataset). Such a mention detection pre-training can potentially result in a less noisy training signal in the early epochs as mentions suggestions for the antecedent matching are already useful. In turn, this can lead to both reduced training time and potentially better results.²

To obtain clusters during inference from antecedent scores, the highest scoring antecedent per mention is selected (or no antecedent at all if the best score is with itself). Then, the suitable mention-antecedent pairs are iterated over: If an antecedent already belongs to a cluster, the mention is added to it. Otherwise, a new cluster consisting (initially) of mention and its antecedent is created.

²The CorefQA system uses additional training data with a separate mention detection training. Wu et al. (2020a) show this provides a performance boost.

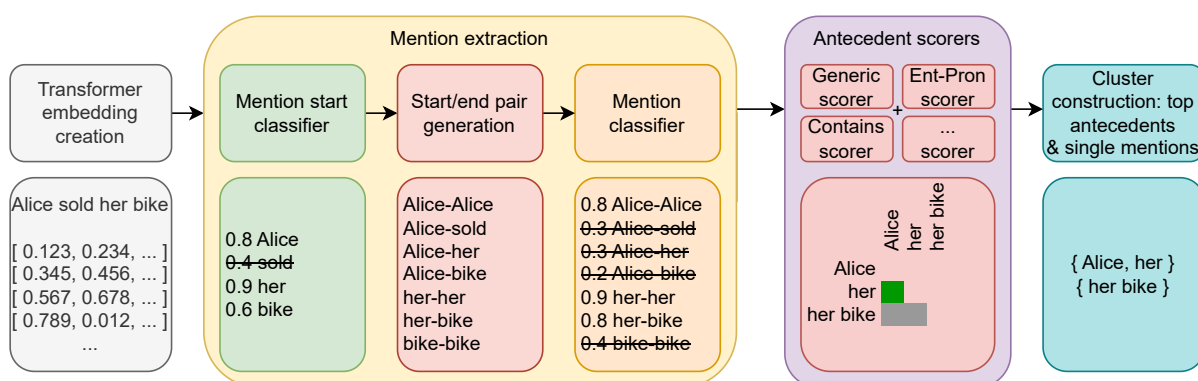


Figure 6.3: Conceptual visualization of our Maverick model architecture processing a tiny example document (“Alice sold her bike”). The model performs mention extraction by finding plausible mention starts, selecting all possible ends, scoring all candidate spans, and finally selects all spans with $p > 0.5$ as mentions. The selected mentions are scored pairwise by multiple antecedent scoring functions depending on the type of pair, e.g. Entity-Pronoun or Pronoun-Pronoun. All pairs with a score $p > 0.5$ are chosen as antecedents, and mentions without any antecedent connection become singletons. Dark green colored squares indicates a good match between mention and its assignment candidate, whereas brighter colors represent a lower score. All values are manually chosen for illustration purposes.

In graph terms, an undirected graph of mentions is created by connecting each mention with its highest-scoring antecedent. In an undirected graph with each node representing a mention, vertices from each mention (that was not filtered out in an earlier step) to its highest scoring antecedent are added. This graph is a graph of entities in which each connected component forms a cluster that represents an entity.

6.4.2 Maverick

Our second model is based on Maverick (Martinelli et al., 2024), a highly-efficient, state-of-the-art coreference resolution model for English. It creates token representations using a transformer encoder, extracts mentions using two classifiers, and finally clusters the mentions by computing the probability of any two mentions belonging to the same cluster by computing the sum of multiple bilinear functions using the mentions’ start and end representations. For a more thorough explanation of the architecture, additionally refer to the original Maverick work by Martinelli et al., 2024.

6.4.2.1 Mention Extraction

While the coarse-to-fine model (Lee et al., 2018) and its derivatives enumerate all spans up to a certain length as possible mentions and select the top k for the mention clustering step, the Maverick system uses a more efficient approach. Mentions are extracted by first identifying

all possible mention starts using a FFNN classifier on each token representation $\mathbf{x}_i$ produced by a transformer encoder for all tokens $t_i, \dots, t_n$ in a document

$$\begin{aligned} F_{start}(x) &= \mathbf{W}'_{start} \cdot \text{LN}(\text{GeLU}(\text{DO}(\mathbf{W}_{start}x))) \\ p_{start}(t_i) &= \sigma(F_{start}(x_i)) \end{aligned}$$

with $\mathbf{W}'_{start}$, $\mathbf{W}_{start}$ being the learnable parameters, and dropout function (DO), GeLU activation, layer norm (LN) and σ the logistic sigmoid function.

For each plausible mention start t_s , *i.e.* tokens with $p_{start}(t_s) > 0.5$, span representations are created for all subsequent tokens t_j until the end of the sentence as the concatenation of x_s , and end, x_j . Another FFNN classifier is used to identify the likely mentions among all candidates

$$\begin{aligned} F_{end}(x) &= \mathbf{W}'_{end} \cdot \text{LN}(\text{GeLU}(\text{DO}(\mathbf{W}_{end}x))) \\ p_{end}(t_j|t_s) &= \sigma(F_{end}([x_s, x_j])) \end{aligned}$$

with $\mathbf{W}'_{end}$, $\mathbf{W}_{end}$ being learnable parameters. All candidate spans having $p_{end}(t_j|t_s) > 0.5$ are selected as the final mentions for the mention clustering step.

6.4.2.2 Mention Clustering

The Maverick model (Martinelli et al., 2024) uses the mention clustering of the LingMess model (Otmazgin et al., 2023), which in turn is an extension of the s2e model (Kirstain et al., 2021). We first briefly describe the mention clustering of the s2e model as implemented in Maverick and our model adoption. A mention $m = (\mathbf{x}_s, \mathbf{x}_e)$, consisting of the start $\mathbf{x}_s$ and end $\mathbf{x}_e$ token hidden states, is transformed into a start and end representation using two fully-connected layers with $\mathbf{W}_s$, $\mathbf{W}'_s$, $\mathbf{W}_e$, $\mathbf{W}'_e$ as the learnable parameters:

$$\begin{aligned} F_s(x) &= \mathbf{W}'_s \cdot \text{LN}(\text{GeLU}(\text{DO}(\mathbf{W}_s x))) \\ F_e(x) &= \mathbf{W}'_e \cdot \text{LN}(\text{GeLU}(\text{DO}(\mathbf{W}_e x))) \end{aligned}$$

The probability p_a of the mention m_i to be an antecedent of mention m_j (in other words: to belong to the same cluster) is computed with a logistic sigmoid function σ over the antecedent logits

$$\begin{aligned} p_a(m_i, m_j) &= \sigma(F_s(x_s) \cdot \mathbf{W}_{ss} \cdot F_s(x_{s'}) + \\ &\quad F_e(x_e) \cdot \mathbf{W}_{ee} \cdot F_e(x_{e'}) + \\ &\quad F_s(x_s) \cdot \mathbf{W}_{se} \cdot F_e(x_{e'}) + \\ &\quad F_e(x_e) \cdot \mathbf{W}_{es} \cdot F_s(x_{s'})) \end{aligned}$$

with $\mathbf{W}_{ss}$, $\mathbf{W}_{ee}$, $\mathbf{W}_{se}$, $\mathbf{W}_{es}$ being four learnable parameter matrices.

The LingMess model (Otmazgin et al., 2023) uses multiple mention-antecedent scorers instead of only the single generic scorer found in the s2e model. The scorers handle different linguistically motivated categories that any pair of mentions m_i and m_j belongs to: PRON-PRON-C, PRON-PRON-NC, ENT-PRON, MATCH, CONTAINS, OTHER. For example, in case m_i is a pronoun and m_j is a proper noun, the ENT-PRON scorer is used to compute p_a . The final

score of two mentions is the sum of the generic, shared scorer (like in the s2e model) and the expert scorer for the specific category. During training, only the shared scorer and the relevant expert scorer receive gradient updates from the loss function.

6.4.2.3 Major Changes from original English Maverick

While we experimented with numerous architectural changes to the model (see Section 6.5.3), there are three major changes in our final model.

First, in contrast to all previous encoder-based end-to-end neural coreference architectures, Maverick uses the binary cross-entropy loss with teacher forcing instead of the complex log-marginalized loss function. The log-marginalized loss function was used due to the computationally required pruning of mention and antecedent candidates. We additionally eliminated the need for teacher forcing by limiting the number of mentions predicted early in training to a computationally feasible amount.

Second, instead of the English DeBERTaV3 (He et al., 2021) transformer model, we use the recent ModernGBERT (Ehrmanntraut et al., 2025) model, a transformer trained from scratch on German data based on the ModernBERT architecture (Warner et al., 2024) that allows highly efficient processing of long input sequences.

Third, the Maverick model on English has six linguistically motivated categories for the mention-antecedent scorers. While we adapted the built-in lists of pronouns to German, we could not use all categories due to the lexical ambiguity of German pronouns. Thus, we combined the categories PRON-PRON-C and PRON-PRON-NC for dealing with (in)compatible pronoun mentions based on gender, number, and animacy into a PRON-PRON category for all pronoun-pronoun pairs.

6.4.3 Model Architecture Comparison

Our coarse-to-fine and Maverick models share the general architecture with a four-step pipeline: (1) Create token embeddings for the input text, (2) identify mentions, (3) find antecedents for mentions, and (4) convert the antecedent matches to a mention clustering. We will compare these four steps individually before discussing overarching architecture aspects. Refer to Figures 6.2 and 6.3 and compare the different building blocks.

Step 1: Embeddings Both model architectures use a transformer-based encoder model to obtain representations for each input token. The weights of the language model are also changed during training via backpropagation. While our coarse-to-fine model uses chunking to slice the input into non-overlapping chunks of at most 512 sub-word tokens split at sentence ends, the Maverick model processes input documents as a whole. The chunking approach makes the processing faster at the cost of a reduced antecedent resolution quality when working on embeddings across different chunks.

Step 2: Mention identification Our models differ significantly in their mention identification step. The coarse-to-fine model generates all possible spans up to a certain length, and scores these candidates. These mention scores cannot be used for classification purposes. Instead of predicting mentions, the coarse-to-fine model limits the candidates to a computationally feasible amount by taking only the top n scoring mentions. In comparison, the Maverick model uses a very efficient mention extraction by first classifying potential mention starts

and then generating all possible start-end pairs up to each sentence end. A mention classifier predicts the final set of mentions to be used in the next processing step.

Step 3: Antecedent matching Whereas the coarse-to-fine model stores mention candidate vector representations in memory limiting the maximal number of candidates, the Maverick model uses implicit representations by working on the start and end token representation of each mention separately. The coarse-to-fine model first performs a fast comparison of all mention candidate pairs before using a more expensive scoring for a fixed number of promising antecedents for each mention. The Maverick model only performs this fast comparison of all mention pairs. However, its scoring actually consists of multiple scorers computed individually.

Step 4: Clustering Both models retrieve the clustering information from the top scoring antecedent of each mention. The coarse-to-fine model cannot find singleton mentions as its mention identification step only creates mention candidates and relies on the antecedent matching step to only retain mentions participating in at least one coreference chain. The Maverick model can predict singletons straightforward: Each predicted mention not participating in a coreference chain becomes a singleton mentions.

Overarching aspects Our coarse-to-fine model implementation makes heavy use of batching to be fast to compute. As it uses fixed limits for the number of mention candidates and antecedents, batching is highly efficient for documents of similar length as the number of mention candidates is identical reducing wasted computations to a minimum. The computationally demanding steps 1–3 operate on a batch of documents in parallel, thereby ensuring effective usage of the GPU (Graphics Processing Unit). Our Maverick implementation processes one document at a time as both the mention extraction and the antecedent scoring step work with greatly varying number of items per document resulting in large computational overhead with batching. Our coarse-to-fine model uses a single loss function on the antecedent scores to train the entire NN, thereby producing uninterpretable mention scores. The Maverick training uses three individual binary cross entropy loss functions summed together: Start-mention loss, mention-classifier loss, and antecedent score loss. This results in interpretable scores allowing to directly classify each item, thereby reducing the amount items considered in subsequent computations.

6.5 Experiments

In this section, we describe evaluation details before we perform preliminary experiments on our two models individually to find the best model design and training procedure. Subsequently, we perform experiments on three datasets introduced in Section 6.3: TüBa-D/Z release 10.0 (TüBa10) and the German part of the SemEval 2010 challenge (SE10) with and without singletons.

6.5.1 Evaluation

Evaluation of coreference data presents a challenge, different proposed metrics emphasize different aspects of a model’s performance. The de facto standard metric for evaluating coreference is the CoNLL-F1 score first used in the CoNLL-2012 task (Pradhan et al., 2012):

Language Model	CoNLL-F1
BERT base, multilingual uncased	74.50
BERT base, multilingual cased	74.60
GBERT base, cased	75.35
GELECTRA base, cased	77.01
GNG_ELECTRA base, uncased	77.86
GBERT large, cased	79.05
GELECTRA large, cased	79.24

Table 6.2: TüBa-D/Z 10 development score of coarse-to-fine models with different language models (using 512 as segment size)

The mean of the MUC (Vilain et al., 1995), B^3 (Bagga and Baldwin, 1998) and $CEAF_{\phi_4}$ (Luo, 2005) F1 scores. While each individual metric has severe issues on its own, their average value has been reliable enough for many years. The LEA metric (Moosavi and Strube, 2016) is a more recent metric that addresses many shortcomings of the above metrics. We evaluate and compare our models using both the CoNLL-F1 and, whenever possible, LEA.

We use the training, development and test splits that are described in Section 6.3. Although the SemEval dataset contains singleton mentions, the coarse-to-fine model cannot predict these as it only predicts clusters consisting of at least two entities. Following the work of Roesiger and Kuhn (2016), singleton mentions are ignored when computing the scores of the model’s predictions. To achieve this, we create a second variant of the SemEval 2010 dataset with all singleton mentions removed during training and testing.

6.5.2 Preliminary Model Design Experiments: Coarse-to-Fine

To find the best coarse-to-fine model for our main experiments, we evaluate different PLMs, their context size and hyperparameters affecting the mention and antecedent scorers.

Pre-trained Language Models We evaluated multiple PLMs for our coarse-to-fine model. As a baseline, we include the multilingual BERT base model (in both the cased and uncased variants) by Devlin et al. (2019). Chan et al. (2020) recently published German BERT and ELECTRA (cased, both base and large) denoted as GBERT / GELECTRA in Table 6.2. In addition, we included another ELECTRA model (uncased, base) by German-NLP-Group denoted as GNG_ELECTRA³. We find that all the recent German language models perform better than the multilingual BERT. For the base models, ELECTRA outperforms BERT by a substantial margin. Using large models, ELECTRA performs marginally better. Based on the results shown in Table 6.2, we selected GNG_ELECTRA as the base and GELECTRA as the large model for our remaining experiments.

ELECTRA Context Size Like Joshi et al. (2019), we split documents into non-overlapping ELECTRA contexts, evaluating different splits for contexts as shown in Table 6.3. While Joshi

³Model description at

<https://huggingface.co/german-nlp-group/electra-base-german-uncased>

Segment Length	F1 (base)	F1 (large)
128	75.69	76.28
256	76.56	77.29
384	77.01	78.51
512	77.50	79.27

Table 6.3: TüBa-D/Z 10 development CoNLL F1 score of coarse-to-fine models GNG_ELECTRA (base) and GELECTRA (large) with different segment lengths.

Foundation	CoNLL-F1
mdeberta-v3-base	76.7
LLaMmleIn2Vec-1B	46.7
ModernGBERT-1B	81.4

Table 6.4: Performance of our Maverick models on the TüBa10 development set using different foundational language models.

et al. (2019) show that for English BERT base/large a segment length of 128/384 is optimal, this does not hold true for our German models and dataset where larger segment lengths perform better. Our results are in line with the intuition that larger context sizes provide more contextual information for any given mention. Thus, we use a segment length of 512 in our coarse-to-fine models.

Hyperparameters In general, parameters affecting computational limits have a large impact, all other parameters that we tested had only limited effect. Parameters controlling the pruning (top_span_ratio, max_top_spans and max_top_antecedents) have a strong negative effect when set too low, due to too aggressive pruning. Higher values increase evaluation scores with quickly diminishing returns; yet strongly increase computation time and memory. To reduce GPU memory usage and computation time, we reduced the size of all FFNNs from 3000 used in previous work to 2048 without seeing distinct score changes on the TüBa-D/Z 10 development set. We also increased the size to 4096, resulting in more memory usage and slower computation, but negligible changes in evaluation performance. Thus, our final coarse-to-fine models have a FFNN width of 2048.

6.5.3 Preliminary Model Design Experiments: Maverick

To find the best Maverick model for our main experiments, we performed a number of preliminary tests to evaluate model variations with foundation language models, teacher forcing, and mention clustering scorers. We consider both model performance and runtime characteristics as we are striving for an efficient and effective model.

Foundational language model We evaluate three language models to replace the English DeBERTa-V3 model (He et al., 2021). As the Maverick model architecture is designed around a language model supporting contexts long enough to process an entire document at once, we were limited to very few options. None of the foundation models for our coarse-to-fine

model are applicable, so a direct comparison is not possible. Instead, we evaluate three models with support for the German language and processing of long contexts. First, we tested the multilingual variant of DeBERTa V3, with promising results of 76.7 CoNLL-F1 for a small, multilingual model — albeit having a slow inference speed. A larger LLM2Vec (BehnamGhader et al., 2024) encoder created from LLaMmle2 (Pfister et al., 2024) was faster but produced subpar results of only 46.7 CoNLL-F1. Finally, the ModernGBERT (Ehrmanntraut et al., 2025) model showed strong results with 81.4 CoNLL-F1 while being fast to compute. Table 6.4 shows a comparison of the CoNLL-F1 score on the TüBa10 development set. Thus, we choose ModernGBERT-1B⁴ as foundation for our Maverick models.

Teacher forcing We compare teaching forcing with standard training on the TüBa10 dataset. While training runs with teaching forcing quickly reduced training loss and increased development scores, training without it eventually but consistently led to slightly better scores on the development set. Thus, we deviate from the original Maverick training procedure and use standard training without any teacher forcing for TüBa10 and SE10.

Attention for mention representation Our Maverick model represents mentions using their start token and end token representation as produced by the language model. We experiment with adding a third component, an attention-weighted average of all tokens that are part of a mention span. In preliminary experiments on the TüBa10 dataset, we see no consistent quality difference on the development set when comparing models with/without our additional span representation. As the additional representations lead to increased computation, as evidenced by slower training and inference, it results in decreased efficiency. Consequently, we keep the pure start and end token representation of mentions without the additional representation in our Maverick models.

Mention clustering scorers We compare a single generic mention clustering scorer with the five specialized scorers in preliminary experiments on the TüBa10 dataset. Using the additional scorers consistently leads to better CoNLL-F1 scores on the development set: ≈ 78 versus ≈ 81 . Thus, we decided to use the stronger, albeit more complex model, as it only requires a small amount of additional compute resources compared to the overall required compute, including the language model.

6.5.4 Main Experiments

We train three models per architecture independently on TüBa10 and SE10. For the latter, we train two variants each: with or without singletons. In each training setup, we use the best model configuration found in our preliminary experiments. For our coarse-to-fine model, we re-use the training settings of Xu and Choi (2020) except for the hyperparameter changes described in Section 6.5.2. For our Maverick model, we use roughly the same hyperparameter settings as Martinelli et al. (2024): Different learning rate of 2×10^{-4} for the head and 1×10^{-5} for the language model, learning rate warm-up over 3 epochs, weight decay of $1e - 3$, and up to 300 epochs limited by early stopping.

Table 6.5 shows the results of our main experiments, the test set F1 scores across all evaluated datasets. On TüBa10, our models outperform the previous SotA by a huge margin

⁴https://huggingface.co/LSX-UniWue/ModernGBERT_1B

	CEAF $_{\phi_4}$	MUC	B ³	CoNLL	LEA
<i>TüBa10</i>					
IMS HotCoref				48.54	
IMS HotCoref + gold mentions				65.76	
Our Coarse-to-fine base	75.33	80.90	75.41	77.21	72.84
Our Coarse-to-fine large	77.09	82.23	77.05	78.79	74.67
Our Maverick	80.32	84.64	80.45	81.80	78.41
<i>SE10 with singletons</i>					
Our Maverick	89.28	79.67	89.81	86.26	82.65
Mention-link-append*				86.40	
<i>SE10 without singletons</i>					
CorZu				45.82	
IMS HotCoref				48.61	
CorZu + gold mentions				58.11	
IMS HotCoref + gold mentions				63.61	
Our Coarse-to-fine base	71.13	76.36	70.14	72.54	66.77
Our Coarse-to-fine large	73.47	77.77	72.14	74.46	68.89
Our Maverick	76.14	80.20	75.37	77.23	72.98
Link-append*				77.80	

* uses additional training data & requires enormous compute resources

Table 6.5: Main experiment results. F1 scores of our models on the test sets of TüBa10 and SE10 (with and without singletons) compared with other models: CorZu (Tuggener and Klenner, 2014), IMS HotCoref DE (Roesiger and Kuhn, 2016), and the mention-link-append & link-append models by Bohnet et al. (2023).

Model	Hardware	TFLOP	Total Seconds	DPS
Coarse-to-fine base	1 Nvidia A40 GPU	4350	29	25.1
Coarse-to-fine large	1 Nvidia A40 GPU	8550	57	12.8
Maverick	1 Nvidia A40 GPU	5700	38	19.1
(Mention-)link-append*	8 Google TPUs-v4	3 960 000	2835	0.25

Table 6.6: Inference compute times of our models on TuBa10 test set (727 documents, 14 586 sentences, 252 652 tokens) as total seconds and documents per second (DPS). TFLOP is the maximal accumulated number of compute operations in TFLOPS when utilizing the hardware fully over the entire inference runtime. *The compute times for the (Mention-)link-append models are extrapolated from the processing times on OntoNotes reported by Bohnet et al. (2023)

(≈ 30 CoNLL F1). In fact, even if the other system is allowed to use gold mentions, our models still outperform them by 10 to 15 CoNLL F1 points. Using ELECTRA large for contextual embeddings in our coarse-to-fine model yields a small improvement over the base model (+1.5 CoNLL F1 / +1.9 LEA F1). Our Maverick model shows an additional performance increase compared to the large coarse-to-fine model of 3 CoNLL-F1 and almost 4 LEA-F1.

On SE10 with singletons, our Maverick model almost reaches the performance of the sequence-to-sequence mention-link-append system (Bohnet et al., 2023) with $\Delta < 0.2$ CoNLL-F1 points. On SE10 without singletons, our models surpasses the non-neural systems again by a wide margin (≈ 27 CoNLL F1) and almost ($\Delta < 0.6$ CoNLL-F1 points) reaches the results of the sequence-to-sequence link-append system by Bohnet et al. (2023). Using a large foundation model for coarse-to-fine yields a small improvement of 1.9 CoNLL and LEA F1 over the base model. Our Maverick model again shows an increase of almost 3 CoNLL-F1 and over 4 LEA-F1 compared to the coarse-to-fine large model.

The sequence-to-sequence models by Bohnet et al. (2023) achieve a miniscule edge in resolution quality over our best models. However, they use additional labeled training data and requires enormous compute resources making it an unfair comparison. Our models require orders of magnitude less compute, making them many times faster. Table 6.6 shows the processing times and required amount of computation to perform inference on the TuBa10 test set. From all evaluated models only our coarse-to-fine base and Maverick are sensible choices for practical application. Our coarse-to-fine base model is slightly faster than our Maverick model at reduced quality. Our coarse-to-fine large model provides a higher quality than the base size but is slower to compute than our Maverick model providing an even higher quality. When comparing our models with the sequence-to-sequence models by Bohnet et al. (2023), it becomes obvious that these models are of less practical relevance given the enormous difference in both compute and speed while achieving similar quality to our Maverick model. While we cannot reliably provide the exact speedup of our models, we estimate a required compute factor between 137 and 547 given the use of our single Nvidia A40 GPU with 150 bfloat16 TFLOPS (Tera Floating Points Operations Per Second) taking 38 seconds on 252 000 tokens versus their 8 TPUs-v4 with 275 bfloat16 TFLOPS per TPU taking 30 minutes inference on 160 000 tokens depending on unknown implementation details such as parallelizability across TPUs. Thus, we see our models as the current state-of-the-art coreference resolution models on German news texts considering quality and efficiency together.

6.5.5 Error Analysis

We perform a manual, qualitative error analysis of our models' outputs using a self-developed analysis tool that compares two files with coreference resolution annotations: A model's prediction and the ground truth annotations. We only consider unseen documents from the TüBa10 test set in our analysis.

Our coarse-to-fine model generally produces accurate coreference links locally and also document-wide for entities with frequent mentions. Figure 6.4 shows coreference annotations on a news article excerpt where the shown output of our coarse-to-fine large model is identical to the human annotation (document score: 89.08 CoNLL-F1). While entity assignment of mentions, identified in both prediction and gold data, is typically correct, missed and added mentions are more frequent errors. We assume that one reason is a contradicting training

[Bahn-Chef]₁ legt Statistik vor. Bisher keine Erklärung für **[das Unglück von [Eschede]**₃**]**₂ **[Frankfurt]**₄ (taz) - Der Eindruck, daß sich die Unfälle bei **[der Bahn]**₅ häuften, sei nur durch die "Berichterstattung der Medien" provoziert, erklärte **[der Vorstandsvorsitzende [der Deutschen Bahn AG]**₅**, Johannes Ludewig (CDU]**₁**),** gestern in **[Frankfurt]**₄. Zum bevorstehenden ersten Jahrestag **[der ICE-Katastrophe von [Eschede]**₃**]**₂ (3. Juni) verwies **[Ludewig]**₁ auf **[die - bahneigene - Statistik]**₆. [...]

Figure 6.4: Excerpt from a TüBa-D/Z 10 test set document (in total 438 tokens), where the shown output of our coarse-to-fine large model is identical to the human annotation (document score: 89.08 CoNLL-F1)

signal, *i.e.* while some mentions are annotated as such in the gold data, others are not because they are singletons or were missed in the annotation process. The added mentions, however, often form entities that could have been missed by the human annotators.

Our Maverick model generally produces highly accurate coreference links both locally and document-wide. Figure 6.5 shows the comparison of a single, rather short document. We find that many errors in the documents are caused by mentions that are not annotated as such in the gold data. As with our coarse-to-fine-model, during training, the Maverick model is punished for those arguably correct predictions, leading to recall issues elsewhere.

As the inter-annotator agreement on the data is unknown, it is difficult to know whether there is room for improvement or if the precision-recall issue originates from disagreement during annotation. We suspect that major improvements may require data curation or active learning (AL) techniques.

6.6 Conclusion

We successfully adapted two coreference resolution model architectures developed for English to German: The coarse-to-fine model and the Maverick model. On the established TüBa-D/Z dataset, we manage to achieve substantial improvements (up to +33.3 CoNLL F1 points) over the previous SotA. Further, our best models are comparable in quality on the SemEval-2010 dataset to the current SotA while being approximately 75 times faster — by models that use additional training data and are impractically slow to compute, even with the required vast computational resources (8 TPUs-v4 for inference, 128 TPUs-v4 for training). In contrast, all of our models are efficient to compute for practical applications on large-scale datasets. Our Maverick model performs strong on both scoring metrics and in manual evaluation. Our smaller coarse-to-fine model performs worse but is even faster to compute for situations where speed and lower memory usage is much more important than quality.

Based on the high-quality predictions of our Maverick model, we are confident that it works well together with our quotation detection and attribution model developed in Chapter 5. Combined, these models allow automatically extracting who said what to whom in German

0 Berliner Vogel stoppt ICE aus Bonn 1
 0 Berliner Vogel 4 stoppt ICE aus Bonn 5 1
 1 Ein Vogel hat am Samstag nachmittag in Berlin den aus Bonn kommenden ICE 846 1 gestoppt.
 1 Ein Vogel 4 hat am Samstag nachmittag in Berlin den aus Bonn 5 kommenden ICE 846 1 gestoppt.
 2 Wie der Bundesgrenzschutz mitteilte, war nach einem betrieblichen Halt im Bahnhof Charlottenburg 2 die Oberleitung gerissen.
 2 Wie der Bundesgrenzschutz mitteilte, war nach einem betrieblichen Halt im Bahnhof Charlottenburg 2 die Oberleitung 6 gerissen.
 3 Nach Angaben der Bahn AG 3 hatte sich ein Vogel in der Oberleitung verhakt.
 3 Nach Angaben der Bahn AG 3 hatte sich ein Vogel 4 in der Oberleitung 6 verhakt.
 4 Weitere Einzelheiten sind noch nicht geklärt.
 4 Weitere Einzelheiten sind noch nicht geklärt.
 5 Im Reiseverkehr kam es laut Bahn AG 3 zu Verspätungen von 30 bis 40 Minuten.
 5 Im Reiseverkehr kam es laut Bahn AG 3 zu Verspätungen von 30 bis 40 Minuten.
 6 Nach Angaben der Bahn AG 3 wurde niemand verletzt.
 6 Nach Angaben der Bahn AG 3 wurde niemand verletzt.
 7 Die Reisenden konnten auf einen anderen ICE umsteigen.
 7 Die Reisenden konnten auf einen anderen ICE umsteigen.
 8 Die Züge zwischen den Berliner Bahnhöfen Zoo und Charlottenburg 2 konnten nur noch eingleisig fahren.
 8 Die Züge zwischen den Berliner Bahnhöfen Zoo und Charlottenburg 2 konnten nur noch eingleisig fahren.
 9 Die Regionalzüge wurden umgeleitet.
 9 Die Regionalzüge wurden umgeleitet.

Figure 6.5: Visualization produced by our analysis tool for a single document from the TüBa10 test comparing the gold annotations with the prediction of our Maverick model. The document is shown in pairs of sentences where the first sentence is from the gold annotations and the second is the system’s prediction. In the gold annotations, multiple mentions of a bird *Vogel* (cluster 4), the city of *Bonn* (cluster 5), and a reference to the overhead wire *Oberleitung* (cluster 6) were missed. Another tiny detail is the first mention of the train station *Charlottenburg* in sentence 2. While the gold annotations only annotated the word *Charlottenburg*, our system’s prediction span the whole noun phrase *Bahnhof Charlottenburg* which is arguably correct since the second mention of it in sentence 8 explicitly refers to the train stations *Berliner Bahnhöfen Zoo und Charlottenburg* and not to the locality of Berlin also called *Charlottenburg*. Figure taken from Petersen-Frey et al. (2025).

news articles. We describe how these models are connected together in Chapter 9 to provide easy-to-use quotation detection and attribution for real-world applications.

6.6.1 Ethical Considerations

Purely relying on ML models to solve a task always introduces room for issues. Automation with ML can enable misuse, although clearly harmful application in the field of coreference resolution are not known to us. Models may exhibit (unknown) biases based on their design or inherent in their training data. The foundation language models used in our work have

been trained on a vast amount of texts that have not been verified manually. However, our models were trained on datasets that were manually annotated and curated (Telljohann et al., 2004). During manual evaluation of our models’ predictions on texts from the datasets as well as new texts, we did not encounter any obvious biases, such as missing or wrong predictions based on gender, *etc.* In any case, downstream usage of our models should consider potential biases in the model whenever operating on the data from our automated approach.

6.6.2 Limitations

We do not see any major limitations in our models for processing news articles. While they might not always produce perfectly correct results according to the human gold labels, they certainly manage to produce highly usable, reasonable results that likely outperform the coreference resolution quality of a non-expert annotator, since annotating coreference chains is a challenging task, even for trained annotators.

Our coarse-to-fine model uses a chunking approach to process documents longer than 512 sub-word tokens. Documents are sliced into chunks shorter than the maximal encoder context length at sentence ends. While it can theoretically process arbitrarily long documents, the polynomial memory and compute requirements without pruning prohibit processing of book-length documents on current hardware. With the pruning strategies implemented, processing long documents becomes computationally feasible, but the quality would degrade drastically once the number of actual mentions exceeds the number of maximal allowed mentions due to pruning.

Our Maverick model is limited by the maximal encoder context length (8192 tokens for ModernGBERT) as we do not use a slicing technique. For documents longer than the intended encoder context length (8192 tokens for ModernGBERT), we rely on the encoder to still generate reasonably useful embeddings. An alternative would be to employ the same chunking as our coarse-to-fine model — albeit with longer chunks. The mention extraction and mention clustering steps are then performed on the concatenated token embeddings of all chunks.

Although our Maverick model is highly efficient computationally compared to most other high-quality coreference resolution models, there still is a limit on the absolute document length due to memory and compute constraints on GPUs. While the mention extraction step scales well with long documents (linear computational complexity), the mention clustering has quadratic runtime and space complexity depending on the number of extracted mentions. As such, it might not be feasible to use our model on whole books, unless mentions are made scarce, *e.g.* by limiting them to only the characters in a story instead of all linguistically possible mentions. To properly process whole books, merging techniques like hierarchical coreference resolution (Gupta et al., 2024) could be combined with a chunking-based variant of our Maverick model. For shorter documents like news articles or even novellas these limitations are not of practical relevance.

7 Scaling Manual Annotations to a Large Corpus

Abstract Scaling manual work in form of span annotations to data sizes infeasible to fully annotate by hand is valuable in many Digital Humanities (DH) and Computational Social Science (CSS) contexts. In this chapter, we present an interactive method to semi-automatically scale manual span annotations to large corpora. We introduce the setting and common need of DH practitioners, review related work and describe the usage of the envisioned application. Then, we explain our interactive method based on contextualized word embeddings (CWEs) and k -nearest neighbors (k -NN) in detail, perform experiments with named-entity recognition (NER) and sentence classification datasets, and conclude our findings.

This chapter is (partly) based on the following publication(s):

Fynn Petersen-Frey, Tim Fischer, Florian Schneider, Isabel Eiser, Gertraud Koch, and Chris Biemann (2023). “From Qualitative to Quantitative Research: Semi-Automatic Annotation Scaling in the Digital Humanities”. In: *Proceedings of the 19th Conference on Natural Language Processing (KONVENS 2023)*. Ed. by Munir Georges, Aaricia Herygers, Annemarie Friedrich, and Benjamin Roth. Ingolstadt, Germany: Association for Computational Linguistics, pp. 52–62. URL: <https://aclanthology.org/2023.konvens-main.5/>.

7.1	Introduction	125
7.2	Related Work	127
7.2.1	Natural Language Processing Methods for few Training Examples . .	127
7.2.2	Existing Annotation Software	128
7.3	Application	128
7.3.1	Requirements	128
7.3.2	Workflow and User Interface	129
7.4	Method	131
7.4.1	Pre-processing	131
7.4.2	Best-matching Structure Selection	131
7.4.3	Interactive, Semi-automatic Annotation Scaling	133
7.4.4	Fully-automatic Annotation Scaling for Quantitative Analysis	134
7.5	Experiments	134
7.5.1	Datasets	135
7.5.2	Semi-automatic Annotation Scaling	135
7.6	Conclusion	137

7.1 Introduction

There is a strong interest in the Digital Humanities (DH) to apply natural language processing (NLP) methods to explore and analyze large textual data corpora. The DH often rely on

Hermeneutic processes and qualitative methodology like grounded theory. Hermeneutic circular processes and theoretical sampling approaches include iterative search, selection, collection, analysis, and interpretation of research data. Grounded theory can be understood as an interactive process where researchers, participants and data construct research together in interaction repeatedly, producing a category system that effectively captures the research problem. It is of great interest to apply the category system to larger datasets for quantitative analysis, but infeasible to do so manually.

Currently, data scientists would need to train a machine learning (ML) model requiring large amounts of training data that have to be created by qualified annotators using to-be-developed annotation guidelines. This is time-consuming, costly, requires ML expertise, and is consequently rarely done in the context of DH projects. Thus, new methods combining human and computer actions are needed to enable research on larger datasets and foster further research in the digital humanities as typical ML approaches are no good fit for most projects. Recent studies (Ostheimer et al., 2021; Koch et al., 2022) in the field of human-computer-interaction have shown fruitful incorporation of human decision-making in ML processes and that human-in-the-loop methods can significantly improve ML models. ML-supported annotation needs the human supervision and refinement to offer useful and accurate alternatives in qualitative data analysis. Strengthening the synergy between humans and machines is a promising direction where both sides are profiting: ML-based annotation is improved by human refinements, whereas automation aids iterative processes of human meaning-making and interpretive research by scaling annotations to enable the analysis of vast materials.

This work targets qualitative researchers who annotate textual data to analyze it in-depth and want to increase their efficiency and/or want to leverage large datasets as quantitative grounding for their hypotheses. We propose a ML-based assistive system leveraging current NLP to ease the annotation task and semi-automatically scale annotations from few manual annotations to a fully-annotated corpus.

After the user has annotated a few text spans with their categories, representatives for each category are utilized for semantic similarity search to suggest relevant text spans with their context. While the user accepts or discards these suggestions, the system adapts to feedback by updating the category representatives instantly after each verified suggestion. Since verifying suggestions is a much faster task than reading and annotating, users can efficiently annotate their documents. The system automatically fine-tunes models to predict higher-quality suggestions and to apply the learned categories to a large document collection with high accuracy, thereby scaling the annotations.

In this chapter, we make several contributions towards a system supporting researchers during qualitative analysis and aids them in scaling their annotations to large corpora:

1. A two-stage method usable without programming or NLP know-how to semi-automatically scale annotations to large-scale corpora by interactively providing adaptive suggestions to automatically annotate large corpora.
2. A user interface (UI) for quick batch validation of suggestions while still displaying the most relevant contextual information.
3. An evaluation of our method with a simulated annotation process on multiple large datasets for sentence- and word-level annotations showing strong scaling capabilities.

Scope Our envisioned application (see Section 7.3) and methods (see Section 7.4) apply to sentence-level (*e.g.* headlines, arbitrary sentences) and token-level (*e.g.* named entities) annotations. Further, our methods are not intended for a MATTER-style annotation process (see Pustejovsky and Stubbs (2012)) but for use in hermeneutic contexts where users move back-and-forth between understanding parts of a text and the whole (the so-called “hermeneutic circle”), continuously build and modify their tagset resp. labels along the way (see Horstmann (2019)). In contrast, the goal of MATTER (Model, Annotate, Train, Test, Evaluate, Revise) cycle is to create a high-quality corpus for training a ML system.

In the remainder of this chapter, we first review related work, describe the application with a typical workflow, explain our method in detail, perform experiments on two datasets, and conclude our findings.

7.2 Related Work

We present related work divided into two categories. First, we examine natural language processing methods applicable for very few training examples. Second, we look into existing software used to annotate texts that also allow for some form of annotation scaling.

7.2.1 Natural Language Processing Methods for few Training Examples

Active learning (Cohn et al., 1996) is a technique to obtain a classifier by soliciting feedback from the user on the most informative sample identified using *e.g.* uncertainty sampling (Lewis and Gale, 1994) which requires a classifier to output certainty scores. AL is often associated with the Human-in-the-loop (Holzinger, 2016) paradigm where human feedback is integrated in the loop of ML development. We see our proposed method as AI-in-the-loop where ML systems assist the workflow of humans who stay in control all the time. In contrast to AL, our system shall assist the user by providing sensible suggestions, but neither disrupts nor dictates their workflow. While this approach might take more time to arrive at the high-quality classifier, in our scenario this is worth the trade-off.

Few-shot classification of named entities is a task relevant to our annotation scaling scenario where a classifier is trained to generalize to new classes after observing only a small number of examples. One research direction explores metric-based approaches, where distances to class representatives are the basis for classification. This task was tackled by Fritzler et al. (2019) with prototypical networks (Snell et al., 2017) which learn a prototype for unseen classes by averaging the representations of the support samples for that class. However, this approach is not made for incrementally increasing size of annotated data, focuses on the classification of unseen classes, and it does not scale well with increasing numbers of examples. Another direction explores prompt-based approaches, where the task of NER is reformulated as closely as possible to the pre-training task of the underlying large language model (LLM). Schick and Schütze (2021) proposed to reformulate the input to a cloze-style question whereas Ma et al. (2022) rephrase the task to a language modeling objective allowing them to re-use and further fine-tune the language modeling head obtained by the pre-training. Both approaches require training and are thus not interactive.

7.2.2 Existing Annotation Software

Qualitative analyses in DH can be supported with software solutions and ML methods addressing annotation, analysis, and interpretation. There are several licensable digital solutions for qualitative analysis available: MAXQDA and ATLAS.ti are two commonly used closed-source, paid solutions for qualitative analysis trying to offer all-in-one-solutions, but do not include a scaling of word-level annotations as described in this chapter. Prodigy is a paid, closed-source annotation software where the workflow is dictated by the AL model. We discuss several other software solutions in Section 3.1, of which LabelStudio (Tkachenko et al., 2020), WebAnno (Yimam et al., 2013, 2014; Eckart de Castilho et al., 2016), Inception (Klie et al., 2018), CodeAnno (Schneider et al., 2023b), LabelSleuth (Shnarch et al., 2022) and ActiveAnno (Wiechmann et al., 2021) offer some form of (semi-)automatic annotation aid in various variations — albeit no scaling of word-level annotations as described in this chapter.

7.3 Application

In this section, we describe an envisioned software for qualitative annotation of text documents that assists users during their annotation process with ML functionality requiring neither programming nor ML knowledge. The assistance comes in two forms: Providing suggestions for semi-automatic annotation and fully automatic annotation to quantitatively analyze large text collections. In the following subsections, we first state the requirements of such an application to be widely useful for DH researchers, and then explain the workflow together with the UI.

7.3.1 Requirements

Throughout our close collaboration between DH and Computational Linguistics groups, we identified four essential requirements for our system (method and UI) to provide the most benefit to the users:

1. *Usable without ML or programming knowledge.* This enables all researchers to benefit from the method. Otherwise, only a small percentage of potential users could actually use the method in their research.
2. *Applicable from small to large-scale document collections containing thousands of documents.* This allows to use the method for a wide range of research questions, corpora and projects.
3. *Instantaneous responses & quick adaptation to the user's feedback.* This greatly improves the user experience. Otherwise, users would stop using the method and return to previously used approach after some time.
4. *Correction of mistakes.* Users must be able to correct system errors and preventing similar mistakes to achieve their desired outcome which has been found highly important for interactive systems like *new/s/leak* (Wiedemann et al., 2018).

These requirements influence both the choice of our employed ML method (see Section 7.4) and the design of our UI that we present in the context of a typical workflow in the next section.

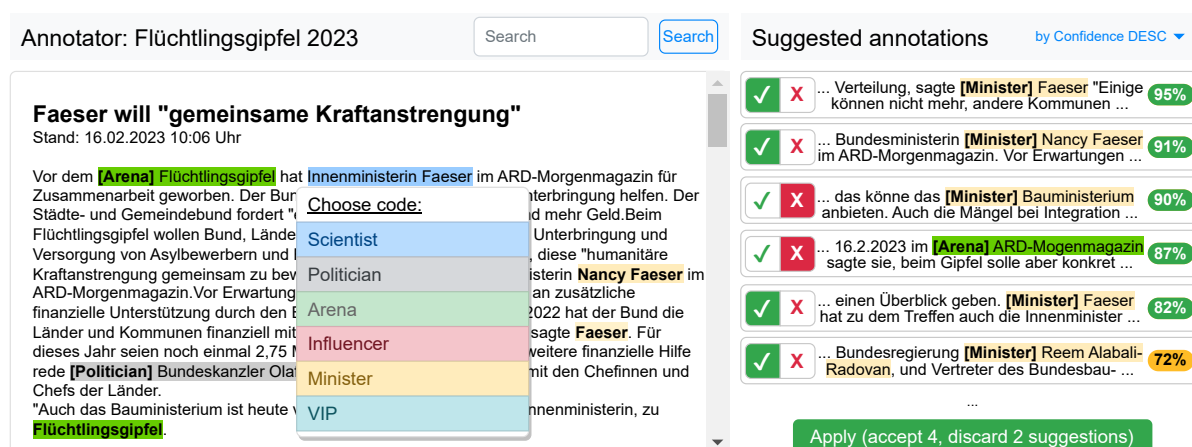


Figure 7.1: Annotation interface. Left: Document with automatic suggestions enabled. Highlighted texts prefixed with [code] are manually created annotations, other highlights are system suggestions. Right: Batch approving/discarding suggestions to semi-automatically scale annotations. Figure taken from Petersen-Frey et al. (2023).

7.3.2 Workflow and User Interface

Imagine Alice, a DH researcher working on a climate project, who needs to identify, categorize and quantify different actors and stakeholders involved in the parliamentary discussions about climate change to answer one of her research questions. She downloaded a large collection of all plenary protocols and printed documents of the German Bundestag which is available as open data. Alice creates a new project in the software and adds the downloaded files.

Manual annotation She works with the software as always: Finding relevant documents by using the built-in search and filtering methods, reading documents, making spontaneous annotations (possibly creating new classes as necessary in this hermeneutic process) while reading and selecting some documents to annotate in detail. During close reading, the system already suggests and highlights text passages in the current document based on the annotated material if the feature is enabled. This feature is shown in the left view of Figure 7.1. Alice gets an idea of what the system would annotate in its current state and might even accept individual highlights to become actual annotations.

Her annotations are quite diverse: Some annotations are named entities coded as actor or stakeholder, possibly hierarchically with fine-grained classes (politician, scientist, activist *etc.*) while others refer to different entities or entire sentences like public statements. After annotating a handful of documents, she has annotated enough different cases of actors and stakeholders to speed up her work with automations.

Semi-automatic annotation scaling She enables the suggestion panel shown on the right in Figure 7.1 to explore annotation suggestions in the text collection. The system provides a list of suggestions showing the context of each annotation, *i.e.* surrounding sentence(s) or the document title. Alice can accept or discard suggestions which are then automatically persisted as annotations in her project. In case the given context is not enough to make a decision, she can view the full document of a specific suggestion to verify it. The suggestion

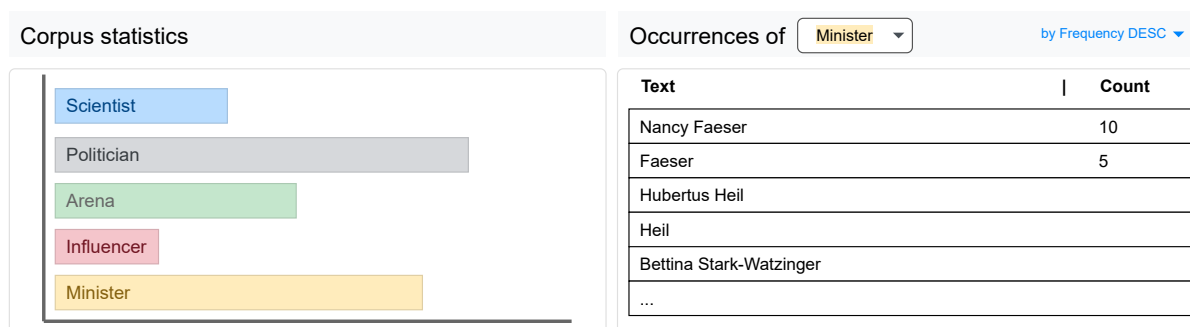


Figure 7.2: Quantitative analysis interface. Left: Estimation of the frequency of annotated categories in the entire corpus. Right: List of all annotations of a selected category. Figure taken from Petersen-Frey et al. (2023).

list can be displayed in two variants for different purposes. As selected in Figure 7.1, individual results from any document are ordered by confidence and shown with a short context. To quickly scale the number of annotated samples, the list shows the right amount of suggestions to fill the display height where the user can accept/reject all entries with a single click as well as select a different choice for each individual item. Alternatively, the list is grouped by documents, ordered by the sum of a documents suggestions' confidence rating, and shows the document title with a link and the paragraph in which each suggestion resides. This allows the user to quickly check if the suggested document is worth reading, *e.g.* being from another field or featuring different aspects or opinion *etc.*

Fully-automatic annotation for quantitative analysis The system indicates to Alice that enough samples have been labeled, and a reliable model is available to apply her current category system to all documents. She double-checks a random selection of these automatic annotations and decides that the quality is good enough for further quantitative analyses. In case she identifies erroneous annotations, she can easily update or remove them. The system adapts automatically and potentially fixes similar errors. Alice can iterate and correct the system as often as necessary. To speed up the process, multiple users can concurrently annotate the corpus and correct the system's suggestions. Finally, Alice retrieves the statistics on the whole corpus such as the frequency of the annotated classes (*e.g.* politicians, scientists *etc.*) or a list of deduplicated occurrences (*e.g.* ministers) sorted by their frequency in the corpus. Figure 7.2 shows the UI providing two connected views for analysis. The left view provides the statistics that allow an estimation of the entire (unseen) corpus to gauge how many instances of which class are present. The right view lists all instances of a specific class sorted by frequency (matched by all occurrences having the same surface text), and allows removal or label-change for each entity. In case there is an instance with a wrongly assigned category in the list, Alice can remove the entry from the list by changing its category, thereby further improving the system's suggestions. By exporting her manual and automatically generated annotations, she can use her favorite analysis and visualization tools to draw further conclusions about her material.

7.4 Method

In this section, we explain our approach to scale few manual annotations with minimal effort to large datasets. We provide a high-level overview before explaining the individual components detailed in the following subsections.

Our method relies on semantic retrieval based on CWEs. It is a continuation of our foundational work presented in Section 4.4 where we retrieved more examples of the same pattern from unlabeled text. We found that it works well with few samples, does not require training, and allows fast prediction. However, it did not reach the quality of a trained model. Yet, CWEs contain a lot of information and make it possible to find other instances of similar patterns! We further established that k -NN is a solid approach for embedding similarity-based retrieval.

In this specialized approach, we initially perform a one-time process to generate CWEs for the document collection, and store the positions of sentences and named-entities in a database. During interactive usage, the user creates some annotations, an algorithm selects the best-matching structure for the text spans, their CWEs are used with customized similarity computations to retrieve and filter candidates, suggestions are provided to the user, the user reviews the suggestions, and the cycle of operations begins anew. For a quantitative analysis, once enough annotations are collected, a k -NN classifier is created and applied to the entire dataset. Optionally, the user refines the classifier by correcting aggregated results.

Before we describe the interactive usage or k -NN classifier creation, we explain the pre-processing and our method to select the best-matching structures that are required for both operation modes.

7.4.1 Pre-processing

All documents added to the system run through an initial, one-time pre-processing phase: Docling (Team, 2024) extracts plain text from any document. Tokenization, sentence-splitting and NER are performed by spaCy (Honnibal et al., 2020). CWEs are computed using SBERT (Reimers and Gurevych, 2019) models for sentences, T-NER (Ushio and Camacho-Collados, 2021) for named entities and RoBERTa (Liu et al., 2019) for other structures (see Section 7.4.2). Multilingual models such as LaBSE (Feng et al., 2022) for sentences, XLM-RoBERTa (Conneau et al., 2020) for tokens could be used to apply our approach across different languages. The embeddings of each structure are stored in an approximate nearest neighbors (ANN) index like HNSW (Malkov and Yashunin, 2018) or FAISS (Johnson et al., 2021) to enable fast retrieval of similar embeddings even for large datasets. The offset-based positions of sentence and named-entities are stored in a database. The pre-processing is executed automatically in the background before interactive use, and never change afterwards, irrespective how many classes are annotated, thereby satisfying requirements 1–3 (4 is not applicable). Pre-computing CWEs of sentences and tokens for the document collection enables instantaneous suggestions.

7.4.2 Best-matching Structure Selection

After the user has made a number of annotations and asks the system to find similar candidates, the best-matching structure for the annotations needs to be found. Each class or label is

assigned to a single structure. The best structure for a class has on average the highest textual span overlap with all already labeled instances of that class.

For all texts annotated, the system has pre-computed the spans for all supported structures, *e.g.* sentences, named entities, noun chunks, n -grams, or single tokens. For each annotation of a class, the token-based overlap is calculated with every span of each structure that has suitable span start and end positions. While some structures always have a guaranteed overlap (sentences or single tokens), because they account for each token in a document, other structures like named entities can have zero overlap with a specific span.

The token-based Levenshtein distance takes this into account: If two spans have identical start and end offsets, the distance is 0. In case the spans overlap to some degree but are not distinct, the number of edit operations is the distance. When two spans have no overlap, they receive a high edit distance because all tokens from one span need to be moved to reach one of the previous cases; thereby producing many edit operations. However, in some cases, *e.g.* the structures are always offset by a few tokens, the non-overlapping matches might actually have a lower edit distance than a structure with overlap that is shorter or longer than the annotation. To guard against selecting a structure that mostly has no overlap at all, structures are excluded when too many non-overlapping matches are selected. This still allows for annotation errors so that a structure is not excluded because a single annotation did not find any overlap.

Sentence-level annotations For annotations that usually span an entire sentence, a full sentence structure is the best match. There are various high-quality sentence representations available of which Sentence-BERT (Reimers and Gurevych, 2019) is selected for multiple reasons: It is a production-ready library, is designed for computational and thus energy efficiency, has a multitude of pre-trained models available, is well documented, allows fine-tuning & customization and is easy to use in general. An advantage of these sentence embeddings is that they are already optimized for cosine similarity search. Thus, the embeddings produced by the Sentence-BERT models use the available embedding space to a great extent and produce meaningful representations because they are trained on supervised paraphrasing data. A further advantage is the low number of embeddings per document: Sentences are rather long compared to single tokens and thus increase the precision and computational efficiency of the k -NN searches applied to the entire embedded corpus.

Entity-level annotations Entity-level annotations are handled explicitly since they are especially interesting as many research questions in DH involve entities. In contrast to sentences, there are no ready-to-use models designed to produce entity embeddings that are compared via cosine similarity. As such a solid baseline is the use of a generic model that creates CWEs for various downstream tasks, *e.g.* RoBERTa (Liu et al., 2019). However, there are many models available that can perform NER and classification: Flair (Akbik et al., 2019), T-NER (Ushio and Camacho-Collados, 2021) *etc.* These models produce embedding representations that allow the classification layer to more easily distinguish between the various classes it was trained on. Since the models produce embeddings for (sub)tokens, but named entities have a varying token length and vector similarity comparisons require a fixed-length vector, a fixed-length representation needs to be created regardless of the number of (sub)tokens. While only taking embeddings of the first subtoken is okay for generally short entities, we average all subtoken CWEs to obtain a single named entity embedding. Alternatively, it is possible to concatenate the first and last subtoken embedding of the span.

Generic span annotations For generic span annotations without specialized vector representations, RoBERTa’s CWEs can be used as a fallback. If annotations are short, *i.e.* mostly single token, the embedding of the first token is chosen as representation. When annotations are longer, a concatenation of the first and last token embedding of each annotation span is used to create the full representation.

7.4.3 Interactive, Semi-automatic Annotation Scaling

Whenever the user asks for suggestions of a specific class c of annotation, the system performs the following steps to produce a list of k suggestions.

Structure selection The system automatically detects which structure fits annotations of class c best by computing and comparing the overlap of annotations with all known structures computed during pre-processing (*e.g.* sentences, named-entities, noun chunks, single tokens or n -grams). Details for this process are described above in Section 7.4.2. This is a fast database operation (< 10 milliseconds) as only offsets for a few manual annotations need to be compared. The structure s with the highest overlap on average is chosen for the next step. For our example, assume the system finds the structure ‘named entity’ fits the category politician best and consequently, the index of named-entity embeddings from the pre-processing is chosen for the next steps.

Candidate retrieval The embeddings of the structures matching each annotation (positive support set, size n) are used to search for the most similar embeddings using cosine similarity in the ANN index. To do so efficiently, a batch query is performed on the index to retrieve a list of candidates for each embedding in the positive support set. In total, $N > n$ similar embeddings and their corresponding text spans are retrieved from the ANN index. Since each individual query returns result lists independently, a single result set is created removing any duplicate candidates.

Candidate filtering Any already annotated candidates are excluded. The remaining candidates are further filtered by removing a candidate if it is nearest to another sample having a class different from c (negative support set). This *advanced* approach often results in large quality improvements (see Section 7.5.2 for a comparison). The *naïve* approach does not filter using a negative support set. Again, a batch query returning only the most similar item is used to do so efficiently. The remaining candidates are re-ranked by their maximum similarity to any annotated sample of class c . Finally, the top k candidates are returned to the user as suggestions.

Reviewing suggestions The user can accept/discard the automatic suggestions batch-wise and accept/discard/edit individual ones. Accepted and edited suggestions are stored as annotations in the database. Rejected suggestions are assigned a hidden ‘not- c ’ class and also stored in the database. The suggestions iteratively improve with every use as more samples of the class c become available as positive support and more samples of other classes become available as negative support.

Requirement check Our method fulfills all requirements (see Section 7.3.1): Requirement (1) is achieved since no configuration is required. Requirement (2) and (3) are fulfilled as the search in the ANN index scales to billions of embeddings and returns results in a few milliseconds regardless of the collection size. In accordance with requirement (4), users can edit/discard suggestions which automatically affects future suggestions.

7.4.4 Fully-automatic Annotation Scaling for Quantitative Analysis

While the interactive mode allows to quickly annotate hundreds of examples, a real classifier is needed to obtain quantitative results on large datasets. To build a classifier, enough negative samples are required beside the positive samples. The classifier relies on the same pre-processing and structure selection as the interactive method to find the best class of annotated samples with their embeddings. If there are fewer negative samples, negative candidates are randomly sampled from all unannotated items (the number of positive samples is usually small compared to all samples). To guard against accidentally choosing a positive sample, candidates are removed if their nearest embedding neighbor belongs to a positive class. This strategy is more beneficial than selecting negative samples by taking the nearest neighbors of known items as it would not produce a diverse negative support set.

A k -NN classifier is constructed from the positive and negative support set. To annotate the entire dataset with this classifier, each unannotated structure s is compared to the support set containing all positive and negative samples. This is efficiently done by computing a single matrix multiplication (for cosine similarity) between the support set and all unannotated items. We considered four different strategies to make the final prediction: (1) *Nearest*: The class of the nearest neighbor is chosen as the prediction. (2) *Centroid*: The positive and negative support set are each averaged to centroid embedding. The class of whichever centroid is closer is chosen. This approach is similar to prototypical networks and computationally highly efficient, but often lacks quality. (3) *Majority voting*: The most frequent class within the k nearest neighbors is chosen as the prediction. (4) *Weighted majority voting*: The similarities of each class of the nearest k neighbors are added and the highest is chosen.

This classification is a fast and quickly adaptable approach since no training is required. A small portion of the annotated documents can be used as a held-out test set to estimate the accuracy of the classifier. In case the estimated quality is not high enough, the systems could warn the user that results may be unreliable and further annotated data is required. Applying the classifier on the corpus allows counting how many annotations of a specific class are in the document collection. A list of all potentially annotated text spans can be produced, merged by the same surface text (*e.g.* all politicians with the same name), and sorted by their frequency. The user can correct the output by assigning a different label to each aggregated group of annotations, thereby immediately improving the k -NN classifier to return updated results in less than a second.

7.5 Experiments

To evaluate our proposed methods, we apply them on fully annotated datasets. This allows us to compare our methods outputs with the correct (as in human created) annotations. We perform two evaluations: (1) We simulate how a user would use the system interactively

Dataset	Type	Size	Classes
OntoNotes 5.0	NER	76,714	CARDINAL, DATE, EVENT, FAC, GPE, LANGUAGE, LAW, LOC, MONEY, NORP, ORDINAL, ORG, PERCENT, PERSON, PRODUCT, QUANTITY, TIME, WORK_OF_ART
Yahoo! Answers	SC	60,000	Society & Culture, Food & Drink, Cars & Transportation, Education & Reference, Science Mathematics, Business & Finance, News & Events, Computers & Internet, Pets, Politics & Government

Table 7.1: Overview of the datasets used in the experiments. In the Type column, SC is short for sentence classification.

and evaluate the quality of the automatic suggestions. In this setting, the goal is to find as many annotations of the desired class with the same manual effort (*i.e.* number of verified suggestions). For named-entity datasets, we directly use the given entity span offsets (instead of first applying an entity detection model). We use the total number of successfully annotated samples as metric. (2) We evaluate the performance of the final k -NN-based classifier created after the simulation with the macro F1 score.

7.5.1 Datasets

In this section, we introduce the datasets and data generation processes used in our experiments. An overview of the datasets is provided in Table 7.1.

OntoNotes5.0 (Weischedel et al., 2013) (ON5) is a well-known dataset of 76,714 samples split into 59,924 *train*, 8,262 *test* and 8,528 *validation* samples. While the dataset provides various word-level annotations, we only consider the named entity annotations. Each sample is a sentence in which each word is labeled as one of 18 NER classes or the other class 0. We created a custom version of ON5 with the same number of samples and splits as the original dataset but only 12 of the 18 classes. Precisely, we removed DATE, TIME, GPE, ORG, ORDINAL, and WORK_OF_ART from ON5 by replacing the respective tags with the 0 tag.

Yahoo! Answers (YA) (Zhang et al., 2015) is a topic classification dataset that includes 4.5 million questions and answers from 10 different categories. We divide the original test set (60,000 items) in half to obtain a validation/test split from which we use the title and its category for our experiments.

7.5.2 Semi-automatic Annotation Scaling

We simulate the usage of the system on our custom OntoNotes 5 and Yahoo! Answers dataset with the following strategy. For each of the unseen classes, we randomly select 20 samples (manual annotation) as the initial positive support set from the validation split and iteratively retrieve ten times 20 suggestions that are accepted/rejected with the human-annotated labels from the dataset. We compare the two approaches explained in Section 7.4.3 to provide suggestions: *Naïve* and *advanced* (using a negative support set). Finally, we build a multi-class k -NN classifier from the collected samples and classify the test split. For the k -NN classifier, we compare four variants nearest neighbor, majority voting ($k = 5$), weighted majority voting

Class	Approach	Samples	(Macro) F1 Score			
			Nearest	Majority	Weighted	Centroid
OntoNotes5.0						
DATE	naïve	161	0.92	0.93	0.93	0.77
DATE	advanced	190	0.89	0.91	0.90	0.67
WORK_OF_ART	naïve	45	0.56	0.61	0.61	0.43
WORK_OF_ART	advanced	89	0.65	0.71	0.71	0.42
ORDINAL	naïve	69	0.91	0.90	0.89	0.43
ORDINAL	advanced	148	0.88	0.88	0.88	0.37
GPE	naïve	200	0.93	0.94	0.94	0.89
GPE	advanced	200	0.93	0.94	0.95	0.89
TIME	naïve	27	0.60	0.62	0.62	0.36
TIME	advanced	99	0.60	0.63	0.62	0.33
ORG	naïve	178	0.86	0.87	0.87	0.87
ORG	advanced	197	0.89	0.89	0.89	0.89
six unseen classes	naïve	680	0.83	0.83	0.83	0.65
six unseen classes	advanced	923	0.83	0.85	0.85	0.62
Yahoo! Answers						
all ten classes	naïve	662	0.50	0.51	0.52	0.53
all ten classes	advanced	1,497	0.56	0.59	0.59	0.58

Table 7.2: Annotation simulation & k -NN classification results. Samples are the number of correct suggestions.

($k = 5$) and centroids. For Ontonotes, we trained a NER model based on T-NER (Ushio and Camacho-Collados, 2021) only on 12 out of 18 classes to produce entity embeddings by simply taking the first token of each entity. To obtain sentence embeddings for Yahoo! Answers, we use the pre-trained S-BERT (Reimers and Gurevych, 2019) model all-MiniLM-L12-v2.¹

The results of the experiments are shown in Table 7.2. While both approaches are able to provide mostly correct suggestions for DATE, GPE and ORG in OntoNotes, the naïve approach has a high error rate for the remaining three classes. The advanced approach provides more than twice the samples for the difficult classes (which are rarer and more overlapping, see Figure 7.3) and reaches an average suggestion precision of 76.9%. On Yahoo! Answers, the advanced approach provides more than twice the number of correct suggestions of the naïve approach resulting in an average hit ratio of 75%.

On OntoNotes, the k -NN classifier produces very strong F1 scores (≈ 0.9) except for the TIME and WORK_OF_ART classes having the fewest samples. While there is only a small performance difference between nearest and (weighted) majority voting k -NN variants, the centroid performs worse by a noticeable margin. The performance on Yahoo! Answers reaches scores of 0.59 with the advanced approach widely outperforming a random baseline of 0.1. We attribute this comparatively lower level to fact that the category in the dataset is not only dependent on the title but also the question text which we did not leverage in our experiments.

¹available on <https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2>

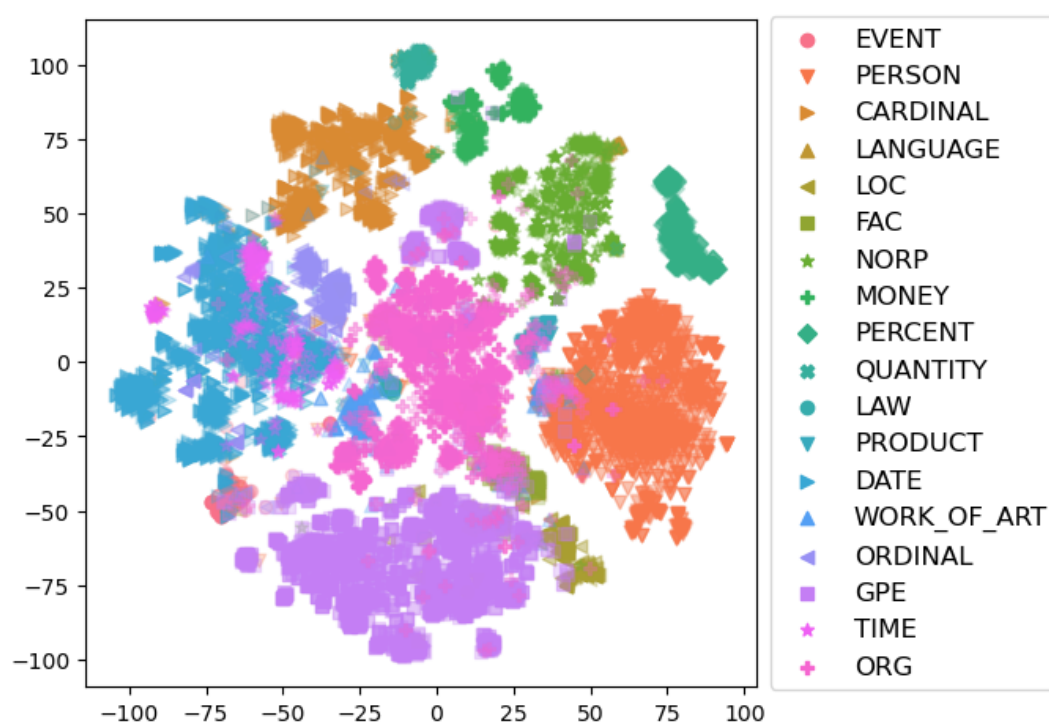


Figure 7.3: t-SNE entity embeddings generated on the ON5 test split using our custom trained T-NER model. Even the unseen classes, DATE, GPE, ORG, WORK_OF_ART, TIME and ORDINAL are visually separated in their own clusters, with TIME and DATE having the most confusions. Figure taken from Petersen-Frey et al. (2023).

We further analyzed the reasons for the behavior of the suggestions and k -NN classifier with a visualization of the entity embeddings in Figure 7.3. Most instances of a class are visually clustered together, even for the six unseen classes. However, DATE and TIME are mixed together and explain the lower scores for TIME as it is the rarer occurring class. Overall, the visualization shows that the use of embedding similarity metrics for suggestions and classification can work well and should also be able to distinguish between subclasses as many of the clusters consist of smaller clusters. We also measured the run-time efficiency of our approach: Generating all suggestions and classifying every entity in the test splits for all experiments took only 0.7 seconds in total.

7.6 Conclusion

In this chapter, we proposed and evaluated methods to semi-automatically scale a few, manual annotations to large corpora by providing interactive suggestions from instantly adaptable classifiers and developed UIs to make our methods usable for domain experts without programming or NLP skills. Our evaluation on existing datasets shows that these methods can quickly scale annotations with minimal manual effort to large corpora to both obtain quantitative results and aggregated lists enabling a verification of the automated processing. Thus, we

see our methods aiding qualitative researchers to bridge the gap to quantitative results and providing quantitative grounding for their hypotheses.

Ethical Considerations Our proposed methods target DH researchers and are intended to assist with qualitative discourse analysis. As with any ML-based method, inappropriate or even misuse is possible. However, we believe including and enabling more researchers to benefit from ML technology outweighs the potential misuse.

Limitations Our proposed methods are certainly not without error. Especially, the fully-automatic annotation scaling has to be used cautiously. We developed this method for researchers to obtain quantitative insights on large datasets. The results are not comparable to a careful, manual or semi-automatic analysis of the full material. Instead, they should be understood as an estimation and help to quantitatively verify hypotheses that emerged from the qualitative analysis. We try to mitigate the issues of the fully-automatic annotation scaling by providing confidence scores where applicable, showing aggregated classification results and allowing the user to correct system mistakes. To evaluate the quality of the automatic annotations, a user can manually annotate a random subset and compare this with the automatic results.

Outlook As our methods are also applicable to image or video annotations via object detection, one might explore to adapt them for a good user experience. In Chapter 9, we show an integration of the developed methods and UIs in production-grade code into our NLP web platform. In doing so, we make our developed methods easily accessible to other researchers and plan to further improve the methods by incorporating more feedback.

Part III

Platform, NLP Method Integration & Conclusion

Abstract In the last part of this dissertation, we develop the open source research platform with a suitable software architecture and implement required functionalities to integrate the natural language processing (NLP) methods presented in Part II, and conclude this dissertation.

In Chapter 8, we perform a requirements analysis to derive the design goals of the platform, and develop suitable architectures for all parts of the platform. In Chapter 9, we highlight selected implementation details, and describe the integration of the NLP methods developed in Part II into the platform including necessary changes and interaction with other functionalities. Finally, we conclude this dissertation and outline future directions in Chapter 10.

8 Platform

Abstract Having developed natural language processing (NLP) methods for Digital Humanities (DH) and Computational Social Science (CSS) contexts in Part II, usable software is needed to enable many DH and CSS scientists to take full advantage of the methods and apply them in their work. We design such a software platform in this chapter by performing a requirements analysis from past project experiences to derive a number of design goals that the platform should follow to be actually used by DH and CSS researchers, and finally develop suitable software architectures on multiple levels of detail to create the foundation for a solid implementation.

This chapter is (partly) based on the following publication(s):

Fynn Petersen-Frey (2023). “Erweiterbare, interaktive Softwareplattform für die Anwendung von Sprachtechnologie in großen Textkorpora zur Unterstützung von Such- und Analyseworkflows in den Digital Humanities”. In: *9. Tagung des Verbands "Digital Humanities im deutschsprachigen Raum" (DHd 2023)*. Ed. by Peer Trilcke, Anna Busch, and Patrick Helling. Trier, Germany & Belval, Luxembourg: DHd 2023 Open Humanities Open Culture. DOI: 10.5281/zenodo.7715444.

Florian Schneider, Tim Fischer, Fynn Petersen-Frey, Isabel Eiser, Gertraud Koch, and Chris Biemann (2023). “The D-WISE Tool Suite: Multi-Modal Machine-Learning-Powered Tools Supporting and Enhancing Digital Discourse Analysis”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Ed. by Danushka Bollegala, Ruihong Huang, and Alan Ritter. Toronto, Canada: Association for Computational Linguistics, pp. 328–335. DOI: 10.18653/v1/2023.acl-demo.31.

8.1	Introduction	142
8.2	Requirements Analysis	142
8.2.1	CLICCS B1 Media Project	142
8.2.2	D-WISE Project	143
8.2.3	HCDS	143
8.2.4	Issues with Existing Software and Methods	144
8.2.5	Common Issues with Research Software	144
8.3	Design Goals	145
8.3.1	Modularity	145
8.3.2	Usability	145
8.3.3	Configurability	146
8.3.4	Widely Applicable	146
8.3.5	Interoperability	146
8.3.6	Large-scale Data Performance	147
8.3.7	Large-scale Data Methodology	147
8.3.8	Interactivity	148
8.3.9	Collaboration	149
8.3.10	Deployment	149
8.4	Architecture	150
8.4.1	Overview	150

8.4.2	Client-Server Communication	151
8.4.3	Server	152
8.4.4	Client	155
8.4.5	Summary	157

8.1 Introduction

Having created fitting NLP methods for DH and CSS in Part II, usable software is needed to enable DH and CSS researchers to apply the methods in their work. Thus, the overall goal of this chapter is to devise a research platform that provides all required basic functionality and allows to integrate the developed methods to deliver their full potential in an interactive research platform. The research platform shall be a versatile tool providing NLP assistance to DH and CSS scientists while also providing commonly used computer assisted qualitative data analysis software (CAQDAS) features typically used in such contexts.

In this chapter, we first perform a requirements analysis to arrive at general design goals. Then, we develop a suitable software architecture that fulfills the design goals. In the next section, we discuss what requirements such a platform in general and individual components or methods should fulfil to be most useful for the work of DH and CSS researchers. In the following section, we devise software design goals from the analysis of the requirements while keeping the overall goal of the research platform in mind. The design goals include both functional and non-functional goals. We discuss all design goals, and describe explicit non-goals. In the last section, we use the design as crucial input to derive the software architecture that follows the design goals and ultimately allows to meet the identified requirements.

8.2 Requirements Analysis

The requirements collected throughout this section build the basis to derive more abstract design goals of the software platform. Any requirements identified in this section are extracted from experiences of numerous collaborations between DH or CSS researchers and NLP researchers during the time as a research assistant and doctoral student at Language Technology (LT) or Hub of Computing and Data Science (HCDS). In particular, this includes experiences (and thus requirements) obtained together with the CLICCS (Climate, Climate Change, and Society) and D-WISE (Digitale Wissensoziologische Diskursanalyse) projects as well as the work with numerous project partners of the HCDS. Finally, common issues with existing software and methods will be summarized in a separate subsection.

8.2.1 CLICCS B1 Media Project

In the CLICCS (Climate, Climate Change, and Society) B1¹ media project, a large collection of news articles permanently collected by the Online Media Monitor² are to be analyzed to answer the overarching research questions which climate futures are possible and which are plausible according to news media. An alternative to MaxQDA (see Section 3.1.2, page 33) was

¹<https://www.cliccs.uni-hamburg.de/research/theme-b/b1.html>

²<https://www.cen.uni-hamburg.de/en/icdc/data/society/omm-mediaanalysis.html>

needed because it can only be used for most purposes of manual single-user work, is ill-suited for multi-user collaboration, and incompatible with large data as it becomes slow, the focus on manual work does not scale, and it provides no overview for large collections of any kind. Yet, a user-friendly software with modern, intuitive user interface (UI) unlike *e.g.* WebAnno (see Section 3.1.1, page 31) was required as the software has to be a tool to be more productive — not less productive or held back. As the software shall be used collaboratively, user and roles management are required.

Required annotation features includes a document-level hierarchical codebook as in *e.g.* CodeAnno (see Section 3.1.1, page 31) with support for arbitrary span-level annotations from a code-set with the option to create new codes on-the-fly without interrupting the annotation process. To deal with large collections such as 100 000 news articles, features for searching and scaling manual annotations are wanted. Texts in German and English need to be processed as these are collected in the Online Media Monitor. Actors are highly important, so features are desired to help identify and analyze them and their interactions. Automatically finding quotations and linking them to actors is helpful for many specific research questions as it substantially reduces the amount of text to work through manually.

8.2.2 D-WISE Project

The partners in the D-WISE (Digitale Wissensoziologische Diskursanalyse) project³ use a workflow that generally follows a hermeneutic circle, an iterative process of selecting more documents, creating new annotations, and analyzing them. Thus, searching and filtering the document collection are essential functionalities. The interactive nature of the research process requires fast response times to be actually interactive without slowing down the thought process. In all features explainability is a crucial property as black-box methods are viewed skeptically in the field. Instead, methods need to be explainable, provide evidence, and allow tuning for the specific task. As a team works on the same research question and data, interactive collaboration on projects is required. So far, the team relied on MaxQDA but dislikes it as it is proprietary, analysis is limited, annotations become unreadable, export (for an external analysis) is severely limited.

During annotation, typically the whole document text is annotated, and becomes unreadable in most tools. The tool should provide an annotator that supports overlapping annotations with a smart coloring scheme. As reflection is central to understanding the process of working on a topic, writing memos and automatically logging performed actions are helpful features. In the D-WISE project only qualitative work is performed, but still in the current age there is an information overflow. Thus, the tool should provide ways to find the right documents to annotate & scale findings to the large unseen collection via (semi-) automatic annotation scaling. An actor network is seen important as the knowledge acquisition considers who said what to whom and who interacts with whom.

8.2.3 HCDS

The Hub of Computing and Data Science (HCDS) requires a generic, undogmatic, widely-applicable software to deal with diverse sets of documents & research questions. The software

³<https://www.dwise.uni-hamburg.de>

should provide high configurability, easy adaptation or modification for specific projects while being simple to get started with the defaults. This allows users to start immediately and use more advanced techniques over time as they feel more confident about using the neural (more block-box) features. It needs an intuitive, self-explanatory UI with inbuilt documentation for any NLP methods available. The installation and deployment must be simple to provide a low entry barrier and leave less room for errors. Since computation is potentially infeasible on client devices and installation on every device troublesome, the software should use a web-based UI while utilizing the available server resources for the heavy lifting and large-scale data storage. To grow and improve the functionality of the software across projects it needs to allow integration of any developed extensions so that more people benefit from the efforts put into it.

Regarding features, in addition to state of the art (SotA) NLP methods, the software also has to provide easy-to-understand basics like word counts, co-occurrences, in what contexts are words of interest used *etc.* In general, widely applicable methods are desired such as topic modeling, semantic search, or quotation detection as these are helpful to work on many research questions in DH or CSS contexts.

8.2.4 Issues with Existing Software and Methods

All tools mentioned in Section 3.1 lack one or more important aspects listed below:

1. Support large datasets: Performance and functionality for handling those data sizes such as overviews or scaling of manual work.
2. Elaborate search features: Full-text search for surface-form matches, similarity-based search for sentences and documents, searching among annotated segments.
3. Near real-time collaboration on the same data, using the same tags or labels.
4. active learning (AL) or AI-in-the-loop for scaling annotations to large data and quality improvements.
5. (Semi-)automatic network creation to work with actors.
6. Intuitive, easy-to-use UI for researchers from DH and CSS without programming experience.
7. Modern, SotA NLP methods that are exchangeable to be kept up-to-date.
8. Undogmatic design to be of use for many researchers and a multiple of different research questions.

Thus, the platform to be developed shall be designed in such a way that all these important aspects can be fulfilled eventually.

8.2.5 Common Issues with Research Software

Typically, software engineering and architecture is neglected in research software. Well-designed software is not necessary to publish papers, time is scarce, and often methods have more impact in the NLP field than actually building usable software. Moreover, there is often no adequate software engineering knowledge available in the NLP research teams. Thus, the goal typically is to build something that only works for the very specific single scenario tested, while cutting corners wherever possible. This approach has multiple negative implications:

1. It is challenging to modify such software because it is not build with change in mind.

2. It usually results in poor user experience as some parts stop working because the user has slightly different data *etc.*
3. It often does not run at all on other systems — unless containerization is properly used.
4. The efficiency and run-time performance is neglected as it is not important for a demonstration.
5. Overall, it only has a very limited fitness for actual use and application.

Thus, the questions arises how to build research software that does not exhibit these issues. In the next section, we define software design goals to prevent such issues with the envisioned platform, and to help create a suitable software architecture for it.

8.3 Design Goals

Having collected numerous specific requirements in the previous section from a diverse set of past collaborations, we now distill and abstract these requirements into software design goals. We group the individual design goals into broader categories in the following subsections. For each group we describe what the design goal includes and explain why it is important.

8.3.1 Modularity

The design goal *modularity* describes a software consisting of multiple (small) modules that function independent of the specific implementation of other modules; the modules are limited in code size to be manageable during development. Following the principle of component encapsulation, this design goal enables to build a software that can be easily extended due to its modular nature. The benefits of such an architecture are manyfold:

First, it is easy to replace a single, well-encapsulated component with a newer, better or simply different component required for a certain task or project. This can just be a newer machine learning (ML) model, but also the specific vector index or relational database used in the platform. This also makes the software more maintainable in case of critical issues with a certain external component by replacing it with an alternative.

Second, it is easy to add another component of the same type, *e.g.* adding a named-entity recognition (NER) model for a new language or another visualization for the same data. Third, it is easy to change or adapt an individual component (within the bounds of its outside interface) independent of all other components. Forth, the development of components can happen asynchronously in parallel by multiple developers. Finally, components developed for one specific project or task can be transferred back to the main code base so that all subsequent projects can benefit.

8.3.2 Usability

The design goal *usability* refers to creating software that is generally easy to use across multiple dimensions. Usability is a crucial and very essential aspect for any software where alternatives exist. Having the most features available but no good way to use them makes users look elsewhere instead of even trying if this means loosing some functionality. This design goal touches nearly every aspect of interaction between the user and the software:

Installation The easiest installation is performing no installation at all. A web-based application has the lowest hurdle to get started across a wide range of operating systems. Providing a ready-to-use instance is thus the goal — however, the software needs to be able to handle many concurrent users.

Interface While command line scripting is incredibly powerful, a proper visual UI is more accessible to non-programmer users and also self-documenting to a large degree. Working with data typically requires visualization of one kind or the other that are better placed in a visual UI.

Batteries included Most components should work out-of-the-box with sane defaults, so users retain the possibility to change settings but can also get started quickly without requiring a deep-dive into *e.g.* installing a ML model for a standard task or algorithmic hyperparameters.

8.3.3 Configurability

The design goal *configurability* describes that users of the software can adapt its behavior on-the-fly to best support their workflow without needing to change the actual source code, rebuild and re-install it. Configurable software allows it to be used in a wide variety of scenarios as users can customize it easily to fit their use case thereby broadening the number of potential users of the software. Configurability also helps to keep the development effort manageable in case of multiple different user groups that need to be supported. Instead of building a custom variant (or even entirely different software) for each user group, all can use the same software and configure it themselves as needed. Configurability also helps with the next design goal of being widely applicable.

8.3.4 Widely Applicable

The software should be *widely applicable* as user groups change and the research questions they want to answer with the help of the software change even faster. Actually changing the software to cater for every case is prohibitively expensive, and even given almost unlimited resources not fast enough. Being widely applicable helps in increasing the chances that future research questions and workflows can also be supported by default. It also allows people from different fields to use the software — even if they do not need every feature.

8.3.5 Interoperability

Interoperability is achieved when the data and information stored within one application can be exported and imported into another software for further processing. Software is rarely used in the void. Typically, there is software that either was or is used together with the same data; often in a pipeline manner where software 1 produces output for software 2 *etc.*; possibly also iteratively needing a tighter integration.

Interoperability is key to get researchers to actually use the software for more than testing purposes. Arguably nobody will spend a significant amount of time with a tool in a work context without being certain to get the enriched or collected data back out and feed it into another software.

There are different levels of interoperability, ranging from import/export in a software-specific format that needs custom transformation to use it another software, over standardized import/export formats requiring no custom conversion to fully integrated and connected applications where the same data is accessible (and potentially even editable) from multiple applications at the same time. While the full integration is very specific to the involved applications and not generally feasible, the design goal of interoperability for a research platform should at least provide basic import/export in semi-standardized formats allowing trivial conversions for other software (such as reordering columns in a CSV (Comma-Separated Values), or converting JSON (JavaScript Object Notation) to CSV while keeping the same information per entry).

8.3.6 Large-scale Data Performance

Most available DH and CSS analysis software does not support large-scale textual data. However, with current web-scale data (or even large retro-digitalization efforts) it becomes essential for many researchers. Large-scale is difficult to specify as a single number; a collection of 1000 tweets (with few words per tweet) is likely not considered large-scale, a collection of 1000 books likely is, whereas a single long book is not considered large-scale data.

We consider anything as large-scale once it becomes infeasible to manually work through the data with a small research team over the timeframe of several months. When also considering the challenges for a software, this typically means that at least of the following two conditions is met:

1. More than 10 000 documents of any lengths: This causes issues for many existing software systems.
2. More than 1 000 000 words in total: It becomes too much to read.

However, this also greatly depends on the functionality offered by the software, *e.g.* providing a search by filename has a different complexity than searching by paraphrases within documents. This design goal focuses purely on the functional aspect: A software can handle large data without issue if the data is processed in a reasonable amount of time, and the software stays usable just as with a small data amount.

In the development of a software, there are multiple potential roadblocks in basic support for large-scale data such as algorithmic complexity, unsuitable software architecture, implementation issues, ill-suited UI. This design goal of basic support for large-scale data is that above four points are considered so that the software platform can work with large-scale data without obvious flaws.

8.3.7 Large-scale Data Methodology

Being able to handle large-scale data from a software and performance perspective is only the first part, actually making use of the data and providing methods to make it accessible to users opens up entirely new ways for researchers and allows tackling research questions otherwise infeasible to pursue. A methodology for working with large-scale data in DH and CSS contexts can be different things. We do not consider methods in this design goal that work with large-scale data without issue but provide no significant benefit over smaller data sets (*e.g.* word counts improve slightly with a larger sample size, but provide no significant gains, and show quickly diminishing returns). While such methods work, they do not solve

the issues large-scale data has for DH and CSS researchers. In this design goal, we instead consider methods that

1. reduce the amount of data to manually work through to a manageable scale,
2. provide options to find relevant samples from a large collection,
3. scale manual work such as classification or annotation to the large, unseen part of the data,

thereby providing the means to gain additional insights from large-scale data otherwise impossible to do in these research projects.

Search is often a central functionality when the amount of data is too large to look at everything. Search in the classic sense can be used to find a specific, relevant document. Moreover, it should also be possible to search for specific, relevant excerpts across all available documents on a paragraph, sentence or even sub-sentence level. While classic search methods are keyword based, even the advanced and highly useful BM25 (see Section 3.2.2, page 39), more recent neural search methods work quite different, and allow for a more semantic search. However, they typically are expensive to compute and behave as a block box. As one part of this design goal we want to provide a useful semantic search that mitigates these typical issues. Generally, automatic pre-processing of the data can be helpful to reduce the amount of manual work by allowing custom filtering, identifying redundant copies *etc.* Finally, an important step for scaling to large data is not only reducing the data to consider but also the other way around: To scale manual work performed on a subset back to the large unseen data. With this design goal, such methods shall be made available to the users.

8.3.8 Interactivity

DH workflows require *interactivity* *e.g.* due to the iterative nature of hermeneutic processes. Working with a software taking long to respond slows down and makes it no joy to use. Farthest from an interactive software is a batch process with specifying what needs to be computed, waiting a significant time, obtaining results, and interpreting the results. Most interactive are real-time games reacting to every user input perceivably just as fast as the real world would. We consider *interactivity* in a research software between these two extremes: Responses to user input should be near-instant (less than 200ms) for frequent actions, quick (less than 1 second) for most other actions, longer actions should provide progress status and allow concurrently continuing with other actions in the software.

Annotating text should be at least as quick as on paper, otherwise it significantly slows down. In an AI-in-the-loop setting, the user should not wait for the machine — otherwise, the user spends more time waiting than working on their research. The levels of responsiveness need to be matched even for large-scale data, otherwise the software becomes infeasible for this use case.

In short, an interactive software should be highly responsive to achieve a good user satisfaction. A marginally stronger method that is orders of magnitudes slower is often not a good fit for interactive workflows. Thus, with this design goal, the default should be responsive methods, optionally providing slower alternatives if required.

8.3.9 Collaboration

Research is often not only performed by a single individual alone. Instead, multiple researchers collaborate to tackle a task together. Any software used in such a setting should thus support collaboration on the same project.

Collaboration can be achieved within a wide range. In the simplest case, all colleagues work independently on the same data — either doing different sub-tasks or slicing the total data in pieces. When one person augments the work of another, it becomes more challenging, *e.g.* when simply sending the current project state by mail to your colleagues, only one can work on the project at the same time. Modern tools for collaboration on text documents allow working at the same time on the same data and changes made by one person are near-instantly visible for everyone else. For DH and CSS research software however, this is usually not the case.

With this design goal we strive to set a level of *collaboration* enabling concurrent collaboration on the same project. Thus, multiple researchers can work on the same project, but it does not necessarily require real-time synchronization of changes to all concurrently open documents. However, reloading any document should show the latest global state. Such a form of concurrent collaboration allows to be faster when splitting work across people. Further, it enables flexible collaboration where *e.g.* one task performed by one person depends on a previous task by another person, *e.g.* annotations made by a person can be used by another when searching or using data-driven methods.

8.3.10 Deployment

Researchers can use a wide variety of devices for their work, using a type of deployment that does not require any installation for each user is thus favorable. There is often no dedicated personnel to manage a server-based installation, thus enabling a hosted service is great. Keeping the deployment straightforward helps with individual installations. To fulfill the *deployment* design goal the software should be available as a web application, and the server is provided as a self-contained software container that can run on most Linux systems.

Web applications are ubiquitous for a good reason as they can be accessed by virtually any somewhat recent device with an internet connection (or at least network connection for air-gapped systems). They can be used on various screen sizes and input methods ranging from desktop systems with multiple monitors, keyboard and mouse, over small notebooks, tablets to comparatively tiny smartphone displays. Even if not all functionality is made available on every device, the ability of partial access from a smartphone can be valuable in certain situations.

Web applications with data stored on a central server has numerous benefits if managed properly. Updates are applied automatically for users, data can be accessed from multiple devices, data is backed up, computations can be performed on the server, data can be easily shared between users, common data can be re-used for multiple projects enabling efficiency gains (*e.g.* indexing a large background corpus, pre-processing documents from a library collection only once). Yet, for users who dislike such centralization or who have critical data it is possible to simply run the server on their local device. Consequently, deployment of the server should be simple for a single-user, local installation.

Summary

In this section, we listed requirements for a versatile, modern DH research platform and derived a number of design goals from these identified requirements. In the next section, we develop a suitable platform architecture adhering to the design goals.

8.4 Architecture

In this section, we derive a system architecture for the platform and its major components based on the design goals from the previous section: Modularity, usability, configurability, widely applicable, interoperability, large-scale data performance, large-scale data methodology, interactivity, collaboration and deployment.

8.4.1 Overview

A suitable architecture for the envisioned research platform is a client-server architecture as shown in Figure 8.1 for multiple reasons: First, the separation between client and server helps to increase the software modularity as both parts can be changed independently as long as the common API (Application Programming Interface) is respected. For example, the UI of the client can be altered easily without touching any parts of the server. Second, having a potentially powerful machine as the server enables better support for large-scale data that would be infeasible on the low-power client machines. Third, while collaboration would be possible with a peer-to-peer network architecture, the desired level of collaboration is easier to achieve in a client-server architecture where the server is the single source of truth. Concurrent modifications can be handled fine without needing to resort to complex consensus protocols.

A client-server architecture has a few potentially conflicting design goals: For *usability*, the client and server version need to match — the user has to keep the client software compatible with the server version. For *deployment*, it potentially requires installation of multiple software systems (client and server). For *interactivity*, a client-server architecture often results in a less-responsive application because the result of user interactions by default become visible after the changes were applied on the server. Network latency can cause slow applications that are significantly worse to work with compared to local-only software.

When the client software is a web-application, the potential issues with the usability and deployment design goals are resolved. The user does not need to install the software at all and the version of client and server are identical all the time since the server serves the matching

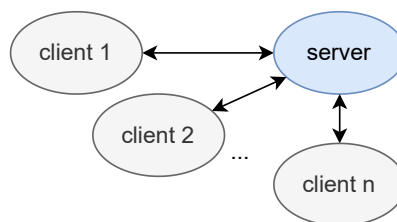


Figure 8.1: Simple client-server architecture

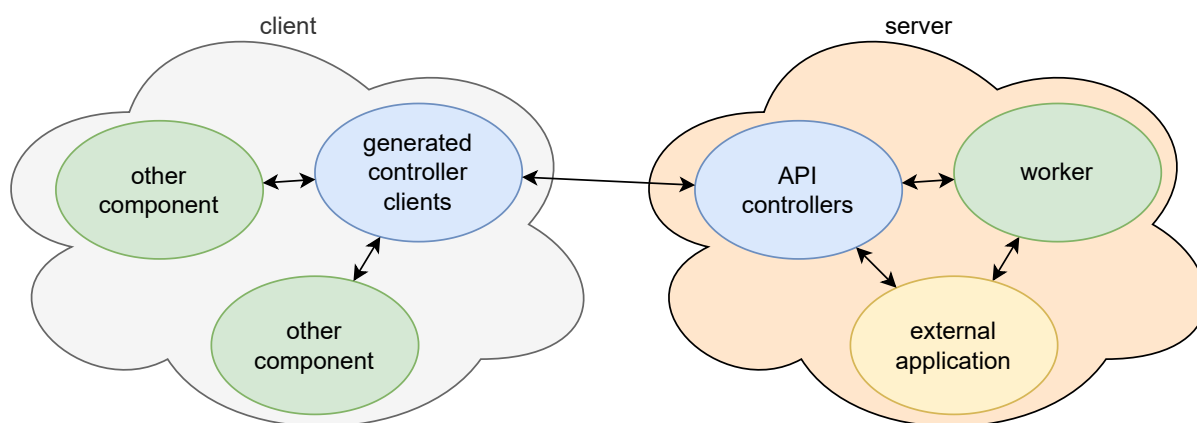


Figure 8.2: Client-server communication architecture. Both client and server use a dedicated component handling the communication between them.

version of the client as a website. The potential issue with interactivity can be resolved with a suitable architecture within the client software relying on optimistic updates.

Consequently, we use a client-server architecture with a web-based client for the research platform. Before describing the inner architecture of the client and server, we first consider the communication between the two parts and elaborate suitable ways to define a common API.

8.4.2 Client-Server Communication

Figure 8.2 shows the high-level communication architecture between client and server, with dedicated components for the client-server communication in both parts.

To achieve the greatest benefits of a client-server architecture, both parts need to be able to be changed independently of the other – while maintaining a compatible API so that they work together as a whole. As the client is a web-based application, there is an obvious choice for the API: A JSON-based REST (REpresentational State Transfer) interface. REST is an architecture style to guide the development of distributed, web-scale systems (Fielding, 2000, pp. 76–105). It emphasizes uniform interfaces, independent deployment of components, the scalability of interactions between them, and creating a layered architecture to enable caching to reduce user-perceived latency. The communication is stateless, so all required information is contained in every request between client and server. This enables the scalability of interactions between them, allowing a server to handle many concurrent requests.

JSON is the natural data format of any web-based application. It allows to serialize any content with a tree-based, nested property-value syntax that can be read and modified natively in the web browser. While there are numerous other data formats (*e.g.* XML (eXtensible Markup Language), protobuf) that could be used, none provide the ease of use and performance of JSON in web browsers. Thus, the communication is JSON-based over HTTP (HyperText Transfer Protocol) that is also used to serve the web application itself (HTML (HyperText Markup Language), JavaScript) from the server.

Having set the communication mode and data format with a JSON-based REST API, the last missing piece is a way to define the actual data to be transferred and specify the supported

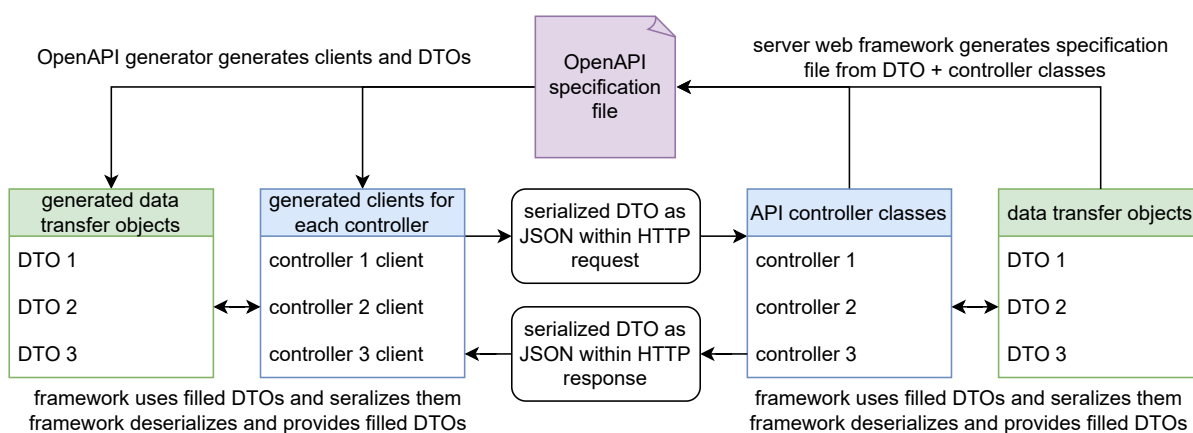


Figure 8.3: OpenAPI workflow. The server controller and DTO (Data Transfer Object) classes declaratively define the specification from which the server web framework generates the OpenAPI specification file. The OpenAPI generator generates clients and DTOs (Data Transfer Objects) in the client programming language.

operations and parameters. For this, we use OpenAPI⁴, possibly the most widely used API description standard. It provides a formal standard for describing HTTP APIs. It is based on a generator pattern, where source code is generated from a specification file. From the same specification file, both the server and client parts responsible for the communication are generated (in different programming languages if needed). Figure 8.3 shows how the OpenAPI specification is created, and used to generate client code. Further, from the machine-readable specification, a human-readable API documentation can be generated providing an overview of the available operations and descriptions of the parameters and results. This setup is aligned with the design goals of modularity and interoperability. First, it enables a clear separation between client and server as both can be developed independently using different programming languages and technologies. Second, the API documentation improves interoperability as it enables an integration with other software as foreign software can interact with the systems via the well-defined API, *e.g.* providing an automated import of data from an archive, library or news scraper.

8.4.3 Server

The server in its overall architecture consists of several components. However, to the client it provides a unified interface in a centralized place regardless of the complexity within the server. This has a number of benefits: It hides complexity from the client which eases client development, and it allows changing server internals freely to evolve independently of client. Any software persisting information needs a kind of database to store the information. For single-process, local programs all required information may be stored in files or an embedded database. For larger server applications with concurrent access a dedicated database service is typically used — which is also the case for the platform developed in this chapter.

Most design goals affect the server architecture. Modularity can be achieved with a multi-service, distributed server with the main service using a layered architecture (external API

⁴<https://www.openapis.org>

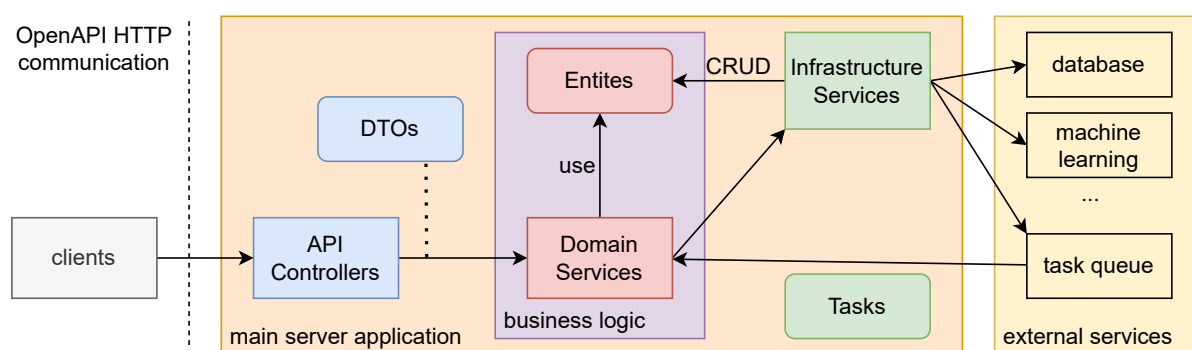


Figure 8.4: Server high-level architecture. The main server application has three layers. It handles client access in API controllers with DTOs to structure information in transit. Business logic uses domain services work with database entities, and rely on infrastructure services to access external components such as database or task queue.

with DTOs (Data Transfer Objects), database entities), business logic in services with proper dependencies, and services as interfaces for external programs (task queue, database *etc.*) to easily swap these out. Configurability means that application settings are taken from environment variables and/or config files, and in-database config values can be changed within the application. Large-scale data performance requires GPU (Graphics Processing Unit) support, parallelization with multiple threads or processes on the same server, and network calls for communication with a task queue for horizontal scaling across multiple machines. Interactivity enforces a highly responsive API server that defers any time-consuming tasks to asynchronous workers. For collaboration a single source of truth is required, concurrency is handled with an atomic database, an event-driven architecture using push instead of polling makes this efficient, and websockets provide real-time updates on the client. Finally, the deployment as connected containers allow for both scaling, easy maintenance and installation.

8.4.3.1 Software Architecture

On a high level, the server consists of two parts: The main server application with mostly self-written source code, and external applications which mostly are ready-to-use services. An overview is provided in Figure 8.4.

The main server application consists of three areas: First, the external API is provided via so-called *controllers* using DTOs to structure data received from or sent to the client.

Second, the *business logic* includes all database entities (all classes relevant for modeling data to be persisted such as users, documents, annotations), and domain services which provide methods to perform complex actions. Domain services work mostly on entities, and do not interact with external applications directly. Instead, they rely on the infrastructure services to perform smaller, more technical operations like storing a file. Domain services orchestrate infrastructure services, and perform custom algorithms on entities *etc.* The infrastructure services provide a high-level interface to external applications in such a way that the implementation of the external service can be changed (*e.g.* using another database software). They provide the methods required by the domain services to *e.g.* upsert an entity to the database, obtain a vector embedding for a text *etc.*

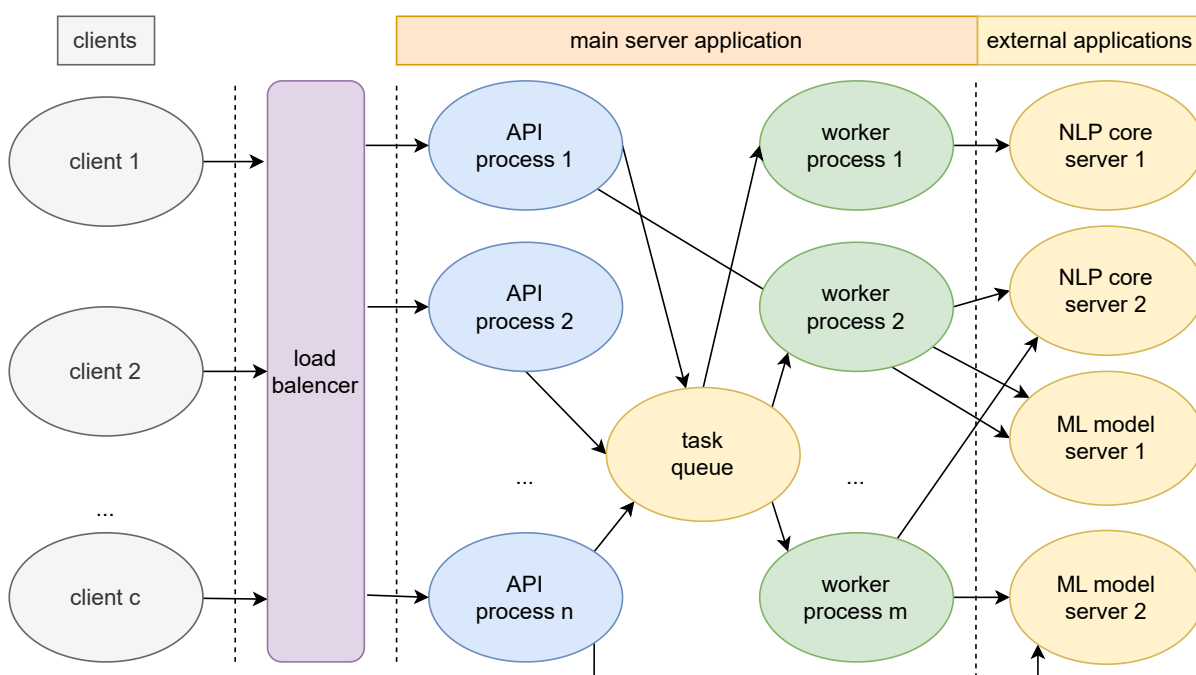


Figure 8.5: Server scaling architecture

Third, external applications are large building blocks providing a specific functionality that is complex and commonly required for many applications, *e.g.* databases, ML inference server or task queue. This also includes custom standalone services, *e.g.* for various NLP functionalities required throughout the platform.

Within this architecture, there are no cyclic dependencies except for one aspect. Since the server needs to handle potentially long-running operations without blocking the API, it uses workers in a task queue. A domain service instructs the task infrastructure service to add a certain task to the task queue, a worker gets notified and processes the task which requires invoking a domain service — that now runs in a worker instead of the critical API process.

8.4.3.2 Scaling Architecture

Handling large and growing data volumes while serving concurrent users is a crucial requirement for the research platform. Processing concurrent clients while performing potentially compute-intensive operations is challenging and requires a non-trivial process architecture. Figure 8.5 shows the developed scaling architecture of the platform. For large deployments, clients do not connect directly to the main server API, instead they connect to a load balancer that performs multiple duties: It caches responses (*e.g.* static assets), handle transport encryption, and forwards requests to an available server API process. The API processes only handle requests that can be answered quickly — *i.e.* which do not involve long-running computations. The API processes use multi-threading and/or asynchronous input/output (I/O) to handle many concurrent requests efficiently. Multiple API processes can run on one machine (*e.g.* 1 per non-uniform memory access (NUMA) node with multi-threading or one per CPU (Central Processing Unit) core with asynchronous I/O). Finally, API processes can be scaled to multiple physical machines.

Requests that require compute or I/O intense operations are deferred to be handled by a worker. Worker and API processes share most of the source code, *i.e.* the domain and infrastructure part, however tasks are only processed by workers which do not use controllers. Worker processes can use multi-threading to utilize more CPU cores if beneficial while keeping memory usage low. While this speeds up the processing of individual tasks, it has some downsides: A crash stops all processing, and there is potential for task starvation where one very long-running operation takes all resources while relatively quick tasks need to wait long. The opposite alternative of using 1 worker process per CPU core with single-threaded processing has other issues: It results in excessive memory usage, does not fully utilize the CPU in case of few concurrent tasks, and is slower in processing a single task. The benefit of individual processes per task is fault tolerance as a crash does only affect 1 task.

The combination of above worker scaling approaches tries to combine their strengths while mitigating the downsides. Multiple worker processes with multi-threading are limited to specific CPU cores allowing for optimal usage of current server hardware (NUMA nodes, complex CPU designs), provide a speedup for individual tasks, and prevent task starvation by dividing tasks into long/short and assigning them to specific workers. Compared to pure multithreading, it results in less processing wasted on improperly scaling algorithms as *e.g.* parallelizing to 128 threads does typically not provide the relative speedup compared to only 8 threads (Amdahl's law, Amdahl (1967)). Moreover, even if large portions are parallelizable, synchronization and locking mechanisms are often not designed for too many threads resulting in degraded performance.

Worker and API processes can run on different physical machines, and can be scaled independently allowing administrators to fine-tune the deployment according to the actual usage based on performance metrics with the potential to automate the scaling. However, for small deployments the best performance is achieved with a single, powerful machine running all components due to the lowest latency. External applications (*e.g.* database) can be scaled independently as it is possible to scale depending on the implementation for stateful applications like the database.

8.4.4 Client

The web-based client is a single-page application (SPA) to provide real-time feedback for user interactions without network latency. Typical downsides of SPAs are irrelevant for the use-case: Search-engine optimization is not important as a separate documentation website can exist, and the slightly longer initial load time is acceptable as it likely opens faster than a locally installed software. The client code consists of two distinct parts: UI-related elements (pages and individual components) and non-UI elements. Figure 8.6 shows the client architecture including selected pages and components as an example.

Pages describe the full view and are navigable by user actions (*e.g.* clicking on the settings button on the main page opens the settings page). UI components are building blocks responsible for a specific functionality, in one of three categories: First, specific custom components used on exactly one page (may directly interact with non-UI elements, *e.g.* to perform API calls). Second, reusable custom components (can be used in various places, all information needs to be passed in from the parent component, may use callbacks to perform actions in a parent component). Third, generic UI components provided by the chosen UI framework.

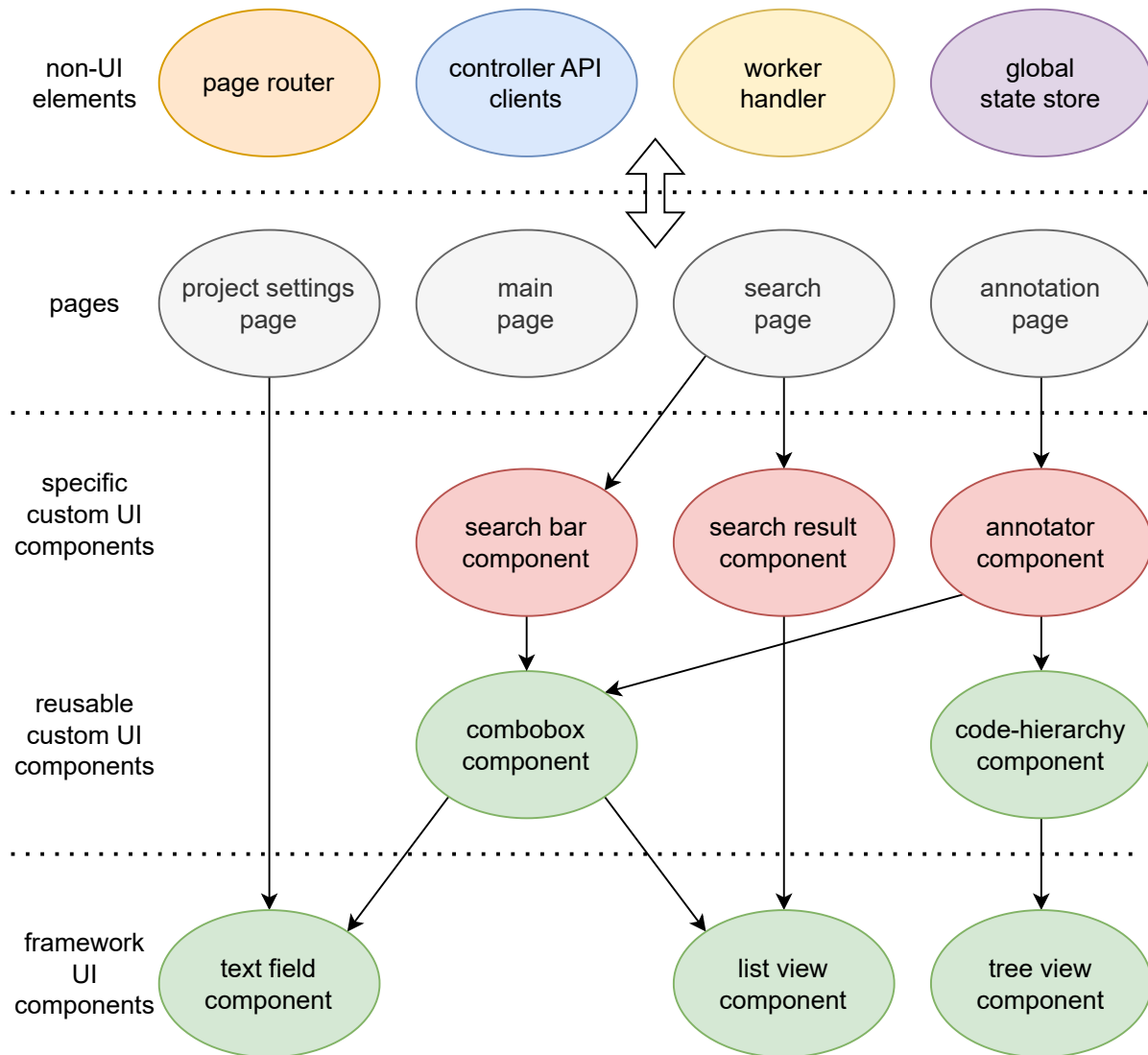


Figure 8.6: Client architecture showing selected pages and components only. The top row shows non-UI elements that can be accessed from the top-most UI layer, the pages. Any UI elements including pages can use other UI components from lower layers only. Below the page layer are specific custom UI components only applicable to a single page. The next layer features reusable UI components created for the platform. The last layer includes generic UI components provided by the used UI framework.

The non-UI elements are the page router (provides method to change the currently visible page), API clients (provide methods to access the server, handles authorization *etc.*), worker handler (provides access to perform certain actions in web workers, *e.g.* compute-intense long-running client operations), global state store (stores global client local state, *e.g.* theme).

Many of the design goals are also relevant for the client: The design goal of *modularity* is also applied within the client with individual components (reusable UI components, local state management), and a modern SPA based on react⁵, vue⁶, or svelte⁷. For *usability*, the interface follows a commonly accepted UI design such as material design, reuses existing components known to users when possible, makes most frequent actions easy with as few clicks as possible, and supports keyboard-use to speed up working with the tool. To provide *configurability*, the UI can be adjusted dynamically as best fit for a task by moving containers or changing their size, and selecting certain preferences in the client settings. To be *widely applicable*, both left-to-right and right-to-left text can be shown, views are compatible with all types of data sizes (many documents, but also long documents) using pagination or infinite scrolling. For *large-scale data performance*, virtual scrolling is used for data displays, and all long processing tasks are handled on the server. To provide *interactivity*, it is a local-first software with changes made asynchronously on the server as enabled by a modern SPA. For concurrent *collaboration*, the client receive updates via websockets and applies any changes automatically to the local state.

8.4.5 Summary

In this section, we derived a number of architectures for the platform on different levels of detail based on the design goals set in the previous section. First, we defined the overall architecture to be a client-server architecture with a web-based client and a server providing a unified, centralized API for the client. Second, we specified the client-server communication using a generated API using OpenAPI. Third, we defined the high-level server architecture. Then, we described how the server can be scaled to process large data volumes and serve many concurrent users. Last, we described a suitable client architecture based on a SPA.

In the next chapter, we describe the implementation of the platform based on the architectures developed in this section.

⁵<https://react.dev>

⁶<https://vuejs.org>

⁷<https://svelte.dev>

9 Applying the NLP methods in the Platform

Abstract Having developed natural language processing (NLP) methods for Digital Humanities (DH) and Computational Social Science (CSS) contexts in Part II, we present an implementation of the platform developed in the previous Chapter 8. We highlight selected implementation aspects required for the integration of the NLP methods, and individually explain the integration of the developed NLP methods into the platform including both frontend and backend changes required.

This chapter is (partly) based on the following publication(s):

Hans Ole Hatzel, Fynn Petersen-Frey, Tim Fischer, and Chris Biemann (2023). “Dimensions of Similarity: Towards Interpretable Dimension-Based Text Similarity”. In: *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*. Ed. by Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu. Vol. 372. Frontiers in Artificial Intelligence and Applications. Kraków, Poland: IOS Press, pp. 1004–1011. DOI: 10.3233/FAIA230372.

Fynn Petersen-Frey, Tim Fischer, Florian Schneider, Isabel Eiser, Gertraud Koch, and Chris Biemann (2023). “From Qualitative to Quantitative Research: Semi-Automatic Annotation Scaling in the Digital Humanities”. In: *Proceedings of the 19th Conference on Natural Language Processing (KONVENS 2023)*. Ed. by Munir Georges, Aaricia Herygers, Annemarie Friedrich, and Benjamin Roth. Ingolstadt, Germany: Association for Computational Linguistics, pp. 52–62. URL: <https://aclanthology.org/2023.konvens-main.5/>.

Florian Schneider, Tim Fischer, Fynn Petersen-Frey, Isabel Eiser, Gertraud Koch, and Chris Biemann (2023). “The D-WISE Tool Suite: Multi-Modal Machine-Learning-Powered Tools Supporting and Enhancing Digital Discourse Analysis”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Ed. by Danushka Bollegala, Ruihong Huang, and Alan Ritter. Toronto, Canada: Association for Computational Linguistics, pp. 328–335. DOI: 10.18653/v1/2023.acl-demo.31.

Fynn Petersen-Frey and Chris Biemann (2024). “Fine-grained Quotation Detection and Attribution in German News Articles”. In: *Proceedings of the 20th Conference on Natural Language Processing (KONVENS 2024)*. Ed. by Pedro Henrique Luz de Araujo, Andreas Baumann, Dagmar Gromann, Brigitte Krenn, Benjamin Roth, and Michael Wiegand. Vienna, Austria: Association for Computational Linguistics, pp. 196–208. URL: <https://aclanthology.org/2024.konvens-main.22/>.

Fynn Petersen-Frey, Hans Ole Hatzel, and Chris Biemann (2025). “Efficient and Effective Coreference Resolution for German”. In: *Proceedings of the 21st Conference on Natural Language Processing (KONVENS 2025): Long and Short Papers*. Ed. by Christian Wartena and Ulrich Heid. Hannover, Germany: HsH Applied Academics, pp. 128–137. URL: <https://aclanthology.org/2025.konvens-1.12/>.

9.1	Introduction	160
9.1.1	DWTS and DATS	160
9.1.2	Contributions	161
9.1.3	Chapter Overview	161
9.2	Implementation Aspects	162
9.2.1	Tech Stack	162
9.2.2	Data Model	162

9.2.3	Services	163
9.2.4	Deployment	163
9.2.5	Annotation Viewer and Efficient Manual Annotation	164
9.2.6	Database Migrations	166
9.2.7	Job System	167
9.3	Semantic Similarity Search	169
9.3.1	Frontend	169
9.3.2	Backend	171
9.4	Coreference Resolution	172
9.4.1	Frontend: Visualizing Coreference Annotations in Documents	172
9.4.2	Backend	172
9.5	Fine-grained Quotation Detection and Attribution	174
9.5.1	Frontend	174
9.5.2	Backend	174
9.5.3	Combining Quotation and Coreference Information	174
9.6	Annotation Scaling	175
9.6.1	Frontend	175
9.6.2	Backend	176

9.1 Introduction

We have developed a suitable software platform in the previous chapter to integrate the NLP methods developed throughout Part II of this dissertation. In this chapter, we apply these NLP methods in the platform by describing the integration into the system as usable software.

We envision a NLP-enhanced research platform that is used by DH, CSS or other researchers. Within the platform, numerous functionalities and automations help researchers to be highly productive and answer their specific research questions. For this, documents are automatically enriched with useful state of the art (SotA) NLP techniques such as named-entity recognition (NER), coreference resolution, quotation detection *etc.* It is possible to filter the document collections with classic text-based and similarity-based search to find documents to look at. An easy-to-use annotator supports long documents while the annotations do not hinder reading the text. Annotation work is speeded up by automatic annotations from NLP models. Further, manual annotations can be semi-automatically scaled up to retrieve answers or validate hypotheses about the unseen part of the document collection. Advanced usage of the similarity search could be applied to find additional relevant material such as opposite opinions on the same topic or same opinion by other authors *etc.*

9.1.1 DWTS and DATS

Implementing and maintaining a software system as large and complex as described in the previous chapter is an endeavor that far exceeds the time available to a single doctoral student. The time and effort required to build the whole system with all required parts including non-NLP-related functionalities such as authentication, authorization, database migrations, data storage, import and export *etc.* would exceed the timeframe available to a dissertation project.

Instead of only implementing self-contained prototypes for the NLP features developed in Part II, a new software platform is developed in a group effort, initially for the D-WISE (Digitale Wissensoziologische Diskursanalyse) project. One outcome of this project is the D-WISE tool suite (DWTS) — in which many aspects of the design goals and architectures from the previous chapter have been implemented. After the D-WISE project ended, the software was rebranded to discourse analysis tool suite (DATS) opening it further to accommodate more use cases. The development of the DATS software has since then been continued within the Hub of Computing and Data Science (HCDS) adding additional developers to contribute code to the software. The NLP methods developed throughout Part II of this dissertation are integrated into DATS.

9.1.2 Contributions

We describe the following implementation contributions to the platform in this chapter:

- A powerful, flexible, connected database model providing the foundation to persist and retrieve information efficiently.
- Database migrations to enable updating instances of the software while keeping any data created.
- A robust, high-performance job system to handle large data sizes, properly utilize the available hardware, and allow flexible integration of new jobs.
- A service architecture with proper interfaces to keep the backend maintainable.
- A text annotator with support for overlapping spans, span groups, intelligent fast code selection, long documents support, smart coloring scheme while keeping text readable.
- Semantic similarity search: For sentences and documents with a plan for a configurable, dimensions-based similarity search.
- German coreference resolution: On-demand job for processing of documents, spans & span groups in the data model and annotation visualizer features.
- Annotation scaling: From qualitative to quantitative analysis with minimal manual effort, suggest similar annotations to the user, and batched suggestion approval for an efficient workflow.
- Quotations: Quotation detection as on-demand job for processing of documents, spans & span groups in the data model and annotation visualizer features

9.1.3 Chapter Overview

In the remaining sections of this chapter, we first present details on the implementation of the platform developed in the previous Chapter 8. We provide an overview of the “tech stack”, *i.e.* the used libraries and external services, before highlighting specific implementation aspects that are highly beneficial to integrate additional NLP functionalities. Afterwards, we individually describe the integration of the NLP methods developed throughout Part II of this dissertation: Similarity search, German coreference resolution, semi-automatic annotation scaling, and quotation detection. Each method is discussed in its own section with a separate explanation of the user interface (UI) and backend functionalities.

9.2 Implementation Aspects

In this section, we provide details on the implementation of selected aspects that are especially helpful for the integration of the developed NLP methods. First, we outline the “tech stack” by providing an overview of the used libraries and external services used in the platform. Second, we explain the data model that builds the foundation for persisting, processing and retrieving any data. Third, present the full service architecture for the platform. Afterwards, we describe selected implementation details of the deployment method of the application, the annotator for fast manual annotation, database migrations, and the job system.

9.2.1 Tech Stack

DATS uses the React¹ framework to create a single-page application (SPA) web-based frontend. The SPA is served with the nginx² web server and its reverse proxy provides access to the backend API (Application Programming Interface).

The backend is written in Python, and consists of three custom components: The API server built with FastAPI³, a worker server for long-running tasks, and a server for machine learning (ML) inference of custom models based on Ray⁴. API and worker share the same source code as most services and entities are used in both. The basic NLP is performed with the spaCy (Honnibal et al., 2020) library in a worker. The backend further includes the relational SQL (Structured Query Language) database PostgreSQL⁵, Elasticsearch⁶ for text indexing, Weaviate⁷ for vector search, Docling⁸ to convert many formats into (plain) text, vLLM⁹ for large language model (LLM) inference, and Redis¹⁰ as a lightweight database for the task queue.

9.2.2 Data Model

One of the most crucial components of the DATS is its underlying data model. The data model connects the business logic elements such as projects, users, memos, or annotations with the heterogeneous documents in their different representations.

DATS stores the majority of its data in a relational SQL database in a fully normalized form. Documents are represented and stored in several ways. The raw files are stored on disk, and can be downloaded by users or read again for computations if required. Semantic vector embedding representations of documents or their segments such as sentences are stored in a separate vector database. Textual information is stored via inverted indices for fast text-based search in a specialized database.

¹<https://react.dev>

²<https://nginx.org>

³<https://fastapi.tiangolo.com>

⁴<https://docs.ray.io>

⁵<https://www.postgresql.org>

⁶<https://www.elastic.co/de/elasticsearch>

⁷<https://weaviate.io>

⁸<https://www.docling.ai>

⁹<https://docs.vllm.ai>

¹⁰<https://redis.io>

The SQL database is the single source of truth. The additional databases for text or vector search merely contain copies or data that can be recomputed for faster access and improved search capabilities.

Within the SQL database, persisted entities are stored as rows in the database tables. Link tables are used to store $n : m$ relations. The entities or tables include project, user, source document, annotation document, span annotation, span text, span group, code, tag, memo, and object handle for the core functionalities. All data (user, documents, annotations *etc.*) is associated with a project. While the source document stores immutable or rarely-changing properties of a document such as name, raw text, date, sentence/token offsets *etc.*, each user can create an annotation document to persist the frequently changing annotation information. A span annotation references a sequence of tokens from a document in form of a span text together with a label from the project-wide code hierarchy. Multiple spans can be grouped together within a span group to *e.g.* represent coreference chains or speaker-quote tuples. Tags are applied directly on documents to flexibly categorize and organize documents in a project. Memos are short documentary texts that can be put on almost any entity via an object handle. The object handle connects any memo-able entity with a memo instance using a typed foreign-key.

Using a normalized relational database enables DATS to provide a highly interconnected access to its data. Changing the code of an annotation in the document annotator immediately impacts all other areas accessing annotations such as the annotation explorer, document search or filtering, code frequency analysis.

9.2.3 Services

Figure 9.1 shows the core services in the backend shared by both API and worker processed together with their connection to the API controller or external applications. Compared to the high-level diagram in Figure 8.4 in the previous chapter, this figure focuses on the services and provides an overview of all required core services and how they interact. Each API controller typically talks to the authentication service (except *e.g.* an optional health-check controller) and exactly one domain service. Each domain service uses one or more infrastructure services to perform any action invoking external applications such as the database or ML inference. Domain services may also use other domain services to provide their functionality. An infrastructure service provides a wrapped interface to the external service to be able to change the implementation easily, *e.g.* file storage, the type of SQL database, or the ML model server. Infrastructure services never use domain services and only interact with exactly one external system.

9.2.4 Deployment

DATS is provided as a browser-based application to users. The deployment of the entire platform is handled with lightweight containers orchestrated with an external container orchestration software such as Docker Compose¹¹. This makes it easy to deploy the entire platform on a personal computer and access the UI locally. Alternatively, the containers are run on a server accessible by multiple users to allow for collaboration.

¹¹<https://docs.docker.com/compose>

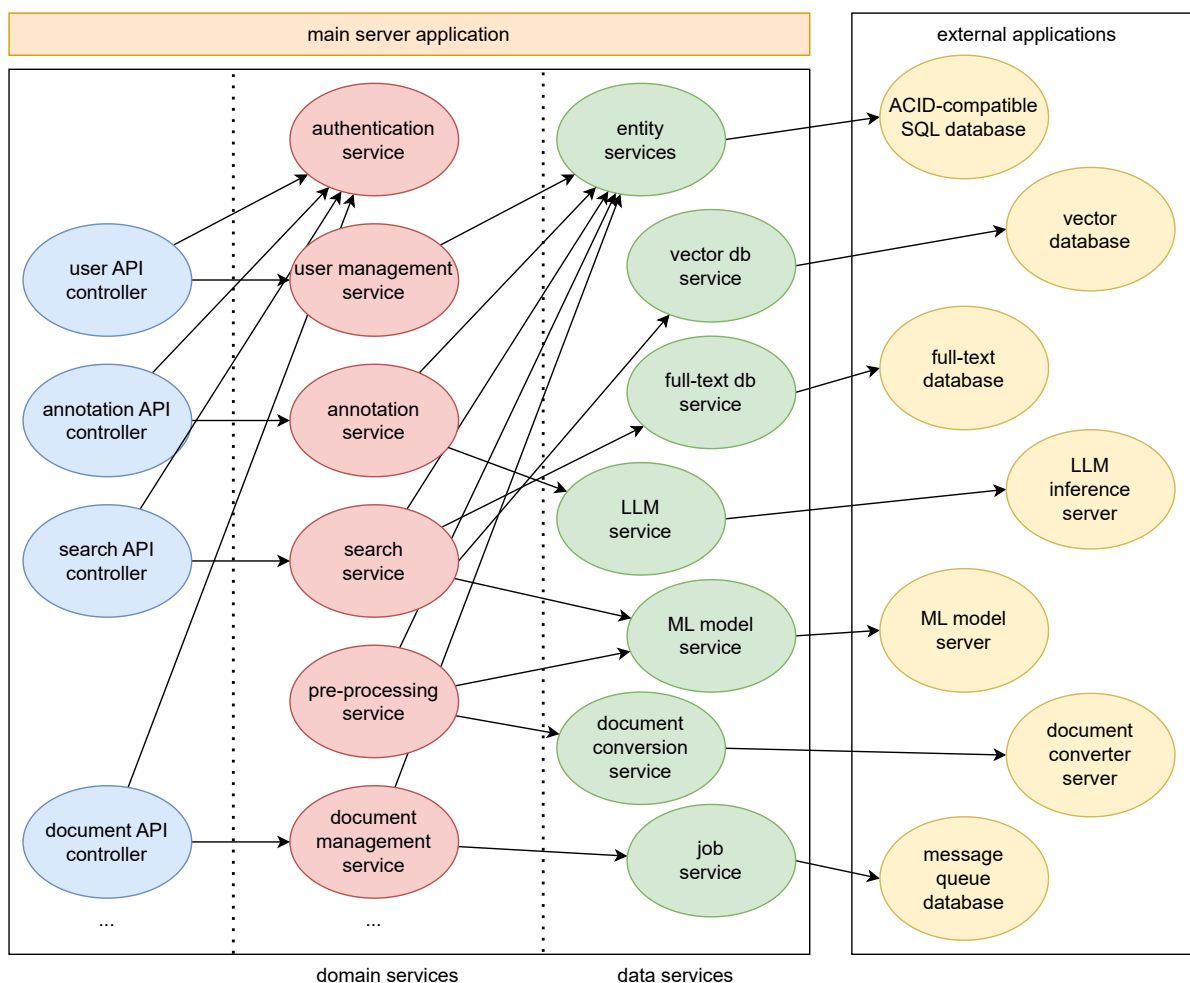


Figure 9.1: Backend services within API and worker code

Figure 9.2 shows the different containers and their interactions. In total, 10 containers are orchestrated as the DATS platform: A combined reverse-proxy and web server proxies requests to the API container, and serves the static frontend files and raw data such as images files. This container is the only accessible entry point from the outside network. The API container handles incoming requests immediately, deals with authentication and authorization, and delegates potentially long-running tasks to the worker via a message queue. Both API and worker container access all four database containers, the ML inference, and LLM inference container. The document converter is only accessed from the background worker.

9.2.5 Annotation Viewer and Efficient Manual Annotation

Annotations are central aspect when working with text in DH or CSS contexts. In contrast to linguistic annotation, the content of a text and its understanding are essential. Thus, the readability of any text must not be impaired by the display of the potentially many annotations. Figure 9.3 shows the annotator as it is integrated into DATS. As an alternative to printing the code name before each span, it is also possible to put the code name above each span at

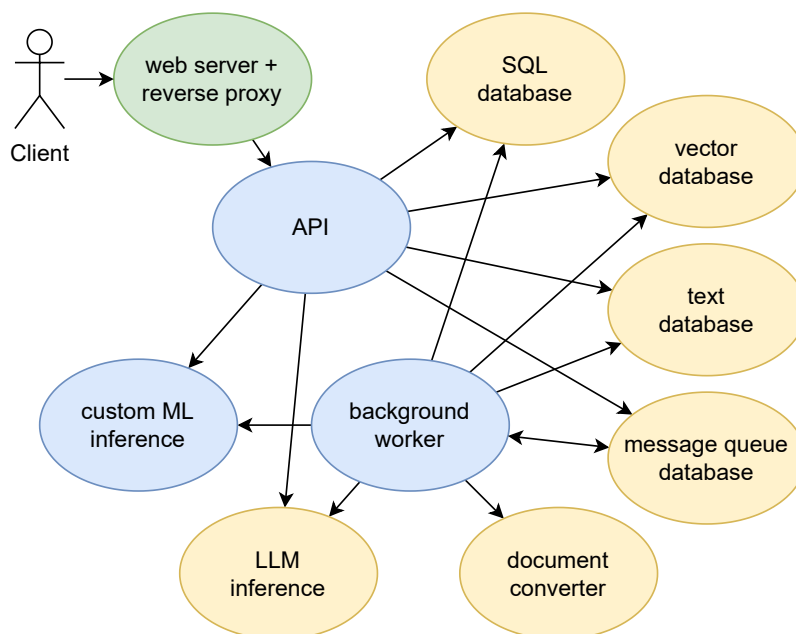


Figure 9.2: Containers of the platform and their interaction. Blue containers run custom code, yellow are externally developed containers, and green is both. The web server serves the specific code for the web SPA.

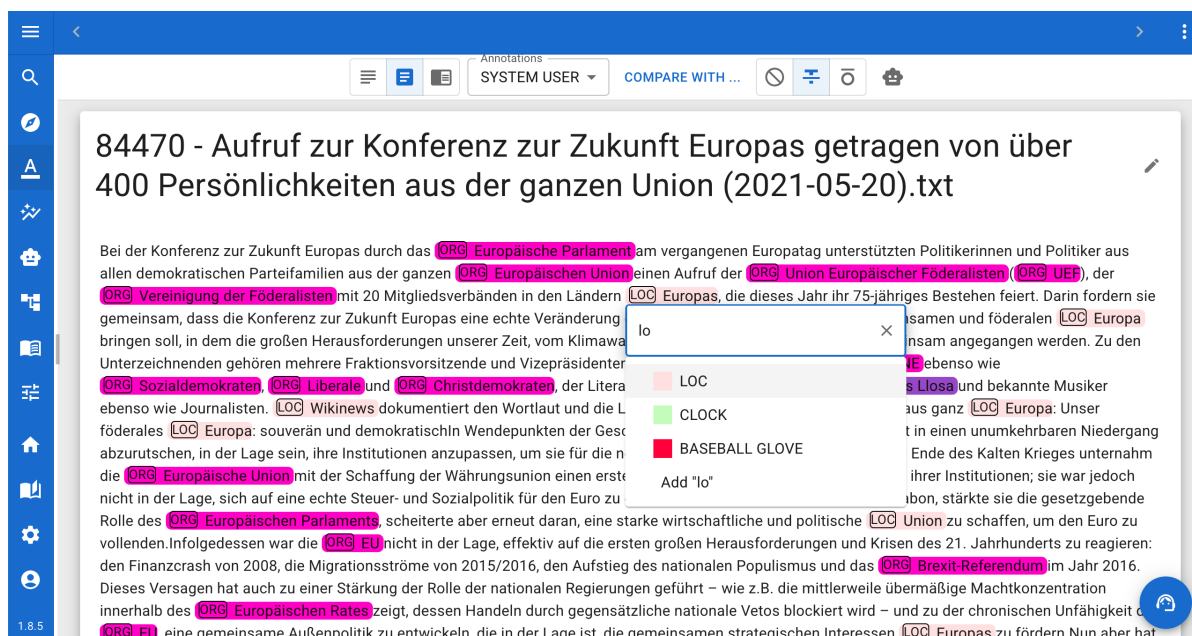


Figure 9.3: Annotator integrated into DATS. The text remains readable while marking all spans accordingly. New annotations can be created by selecting a text span and choosing the desired code via an overlay with autocomplete text input. It is also possible to add new codes on-the-fly (Add "lo").

the cost of an increased screen space usage, *i.e.* showing less text at once. The text remains readable with annotations, and annotations can be intuitively added by selecting a text span or edited by clicking on an existing annotation. Annotations with specific codes can be hidden to further improve readability or focus on selected codes. Codes for annotations can either be selected from the project's code tree or created on-the-fly by entering a new code name in the code selector overlay.

The annotator works on a token level, so that the smallest annotation span is a single token. Overlapping annotations are supported by highlighting tokens belonging to multiple spans with the set of different colors. The annotator is highly interactive as the rendering and necessary computations are performed in the web client, thereby eliminating any latency that would be introduced by server-side processing. To do so, the client uses an *optimistic update* pattern. It immediately performs an action locally, sends the same action to the server assuming a successful processing, and only rolls back the local changes in case the server unexpectedly replies with an error or a network timeout occurs. Thereby, the interaction with the annotator is always instantaneous while keeping the benefits of a client-server architecture such as persistent data storage and collaboration.

As colors are used to visually distinguish the different codes of the annotations, special care was taken to design the color palette. The colors were generated once upfront with a brute force algorithm to have the following two properties: First, colors adhere to the web accessibility standard for readability of black text on colored background¹². Second, color n in the palette is most visually distant from all previous colors in the palette when viewed on a monitor. The algorithm generates all $256^3 = 16\,777\,216$ possible web colors, and filters out all colors that do not meet the contrast criteria to black text or white background. The remaining set of colors is used to find 256 colors starting from a randomly selected color by finding the color with the maximum distance to all selected colors. The perceptually uniform color space $L^*a^*b^*$ was used to compute the distance between two colors. While this approach does not place the colors with optimal distance when using all 256 colors, it has the nice property that it provides more visual distance between the colors when fewer than 256 colors are selected sequentially from the set. As the annotator typically shows far fewer colors, this property is quite valuable. However, since codes have a static color selected from the generated color palette for easy identification across documents, this property is most useful if all codes within the project are also in use in a single document.

9.2.6 Database Migrations

Any software running in a production environment with actual users needs database migrations to be able to make changes to the software to incorporate new features or fix bugs that require changes to the database model. Otherwise, making changes to the database schema would usually result in total data loss requiring users to start from scratch. DATS has a Python-based server and uses SQLAlchemy¹³ to define its entities for the relational SQL database. To easily create database migration semi-automatically, the compatible database migration tool Alembic¹⁴ was integrated.

¹²<https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html>

¹³<https://www.sqlalchemy.org>

¹⁴<https://alembic.sqlalchemy.org>

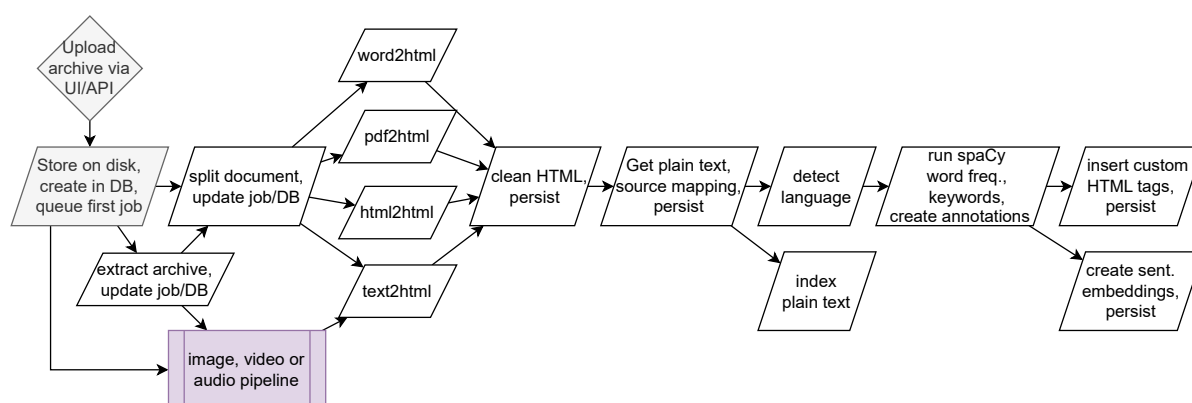


Figure 9.4: DATS text processing flow. Grey depicts actions performed synchronously in the API, purple is a placeholder for another flow, and white depicts jobs relevant to text processing that are performed in a worker.

During development the automatic migration generator compares the current state of the database schema with the specification written in the Python entities. It generates executable Python scripts that transform the current database schema into the specified schema. In case new columns are added or reworked, the generated scripts might need manual adaptation to compute the values of the new columns for existing rows *etc.*

Database migration scripts are executed in ascending sequence during the start-up of the API server. The current database schema version is stored within the database itself, so the system skips any already applied migrations. While DATS has additional databases for text and vector indexing, these databases do not require migrations as they can be recomputed from the SQL database state by re-running the corresponding jobs with the job system.

9.2.7 Job System

As scalability is an essential requirement for DATS, the system was carefully developed to handle large and growing amounts of data and many simultaneous users. To guarantee fast response times of the API under load, potentially expensive tasks are delegated to be processed asynchronously within a worker process using a job system. Tasks are added to the back of a task queue and worker processes take and handle tasks from the front of the queue.

Figure 9.4 shows the mandatory steps of processing a single file containing text. This initial preprocessing enables the core DATS features, making it a central, indispensable component. After a file is uploaded via the API, it is stored on disk, an entry of its existence is created in the database, and the first job is enqueued depending on the file type. For textual files, the *split document* job is enqueued. If the file is an archive, the archive extraction job is enqueued, which in turn enqueues the *split document* job for each extracted file. Image, video or audio files are handled similarly with a specific jobs for each type. As images, audio and video inputs are converted into text eventually, their flows finally enqueue the *text2html* job and thus also run the majority of the text processing jobs.

After *split document*, the raw document is converted to HTML (HyperText Markup Language) with the correct job depending on the type of text, the HTML is cleaned, and finally persisted in the database. Since HTML contains formatting for viewing the document, plain text is

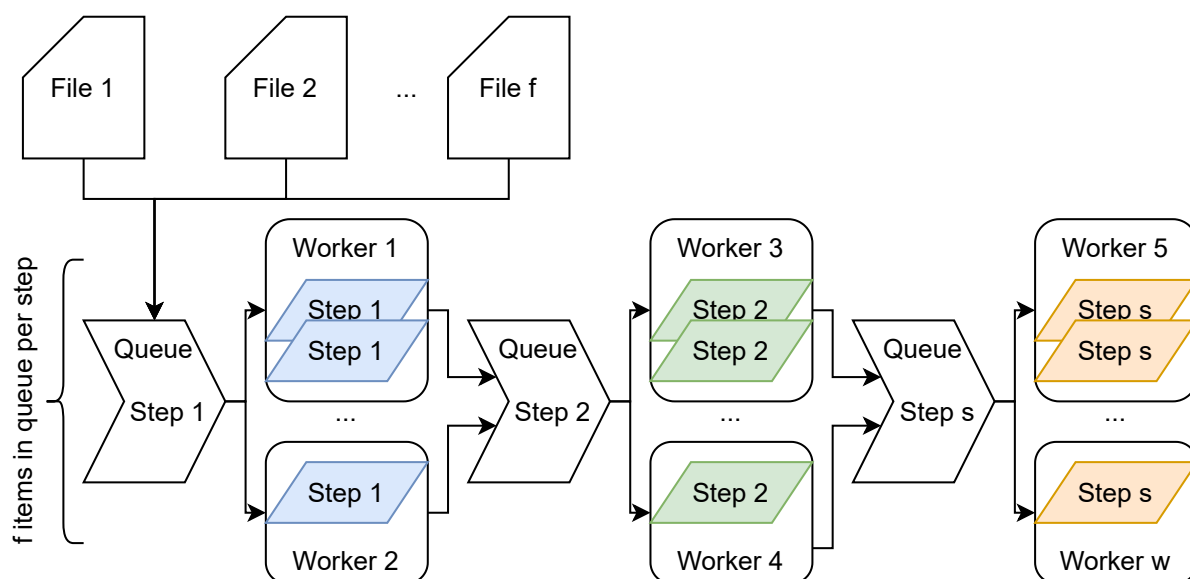


Figure 9.5: Job processing: Queue per step type, batching within queue, parallelism within queue, parallel queues

extracted for processing with ML models together with an offset-based mapping between plain text and HTML. The plain text is processed in parallel by two follow-up jobs: It is indexed in the inverted index, and its language is detected for further processing. After *detect language*, spaCy performs basic NLP tasks such as tokenization, sentence splitting and NER. The tokens are then used to compute word frequencies and extract keywords. The generated information is stored in the database by updating the document and creating span annotations for any detected named entities. Finally, the output is processed in parallel by creating and persisting sentence embeddings, and generating a custom token-aware HTML with custom tags for display in the annotation viewer.

Figure 9.5 shows the highly-parallelized processing of multiple documents. The processing of a single input document is divided into several steps as explained above. While some steps must be executed sequentially because step y depends on the output of step x , other steps can be executed in parallel. The job processing puts the tasks for the steps of each file in a queue. Finishing a step triggers the job system to enqueue the correct follow-up task(s) if any are specified. Thus, it is not required to know the exact steps to compute upfront. Instead, the jobs are dynamically created as needed for the processing.

The different types of jobs have varying compute requirements. Most jobs are run locally in worker container on a CPU (Central Processing Unit). Other jobs require a GPU (Graphics Processing Unit) to be available to the worker. Last, some jobs do not perform any compute at all – instead, they merely wait for the response of network request of an expensive operation performed on another machine. Thus, we differentiate between three job types (CPU, GPU, external API) by having three different queues to best utilize the available hardware. The number of workers per queue can be configured independently to adapt to the machine(s) running the workers.

Together, a high level of parallelism is achieved as (1) some steps for a single documents are handled in parallel, (2) multiple documents can start their processing flow in parallel, (3) there is no practically relevant limit on the queue length allowing tasks to accumulate during busy moments so that workers can always process as many tasks as they can until all work is done, and (4) GPUs or external services have their separate workers and queues to be utilized at maximum capacity at all times — as long as there are tasks to be performed.

As jobs are being used for both interactive (e.g. LLM interaction) and background batch-style tasks (e.g. processing flow during initial file import), the job system must handle usually short-lived interactive jobs before typically long-running background jobs. Otherwise, interactive tasks could be delayed for minutes during a mass import or even hours during parallel ML model training. The underlying task queue implementation RQ¹⁵ of DATS does not support native priority queues as it is complex in a distributed system based on message passing. Thus, to solve the issue of interactive task starvation, the three queues for each processing type are instantiated three times each with different priorities: Low, default and high. Workers always take a task from the highest priority queue before checking for tasks in the next lower priority queue. Thus, interactive high-priority tasks are handled first, unspecified tasks have medium priority, and known low-priority background tasks are handled last.

The job system builds the foundation to efficiently and reliably integrate other NLP models in addition to the basic NLP performed during the import of a file. These models are not applied automatically during import as they require substantially more computational resources while not being needed all the time. Instead, processing all yet unprocessed documents for a selected NLP model can be triggered manually in the UI when needed. In the remaining sections of this chapter, we describe how the advanced NLP models and methods developed during Part II are integrated into DATS, all of them using the job system for long-running background processing tasks.

9.3 Semantic Similarity Search

In this section, we describe the integration of a semantic similarity search inspired by the work in Chapter 4. Although the dimensions-based configurable similarity search has not been implemented yet, we explain how it is planned to be implemented in the platform in addition to the already implemented similarity search.

9.3.1 Frontend

Sentence similarity search The UI provides two options to start a vector-based sentence similarity search. When reading a document, any sentence can be selected by clicking on it and ask the system to list similar sentences (see Figure 9.6). Alternatively, any text can be searched for by using the free-form text input in the search bar (see top bar in Figure 9.7). In both cases the results are displayed in a list ordered by decreasing similarity to the query (see Figure 9.7). By clicking on a result, the full document is opened in another tab focusing on the specific sentence.

¹⁵<https://python-rq.org>

84470 - Aufruf zur Konferenz zur Zukunft Europas getragen von über 400 Persönlichkeiten aus der ganzen Union (2021-05-20).txt

Bei der Konferenz zur Zukunft Europas durch das Europäische Parlament am vergangenen Europatag unterstützten Politikerinnen und Politiker aus allen demokratischen Parteifamilien aus der ganzen Europäischen Union einen Aufruf der Union Europäischer Föderalisten (UEF), der Vereinigung der Föderalisten mit 20 Mitgliedsverbänden in den Ländern Europas, die dieses Jahr ihr 75-jähriges Bestehen feiert. Darin fordern sie gemeinsam, dass die Konferenz zur Zukunft Europas eine echte Veränderung der EU hin zu einem starken, gemeinsamen und föderalen Europa bringen soll, in dem die großen Herausforderungen unserer Zeit, vom Klimawandel bis zur Sozialen Gerechtigkeit, gemeinsam angegangen werden. Zu den Unterzeichnenden gehören mehrere Fraktionsvorsitzende und Vizepräsidenten des Europaparlaments, GRÜNE ebenso wie Sozialdemokraten, Liberale und Christdemokraten, der Literaturnobelpreisträger Mario Vargas Llosa und bekannte Musiker ebenso wie Journalisten. Wikinews dokumentiert den Wortlaut und die Liste der hochrangigen Unterzeichnenden aus ganz Europa: Unser föderales Europa: souverän und demokratisch. In Wendepunkten der Geschichte müssen Gemeinschaften, um nicht in einen unumkehrbaren Niedergang abzurutschen, in der Lage sein, ihre Institutionen anzupassen, um sie für die neuen Gegebenheiten zu rüsten. Nach dem Ende des Kalten Krieges unternahm die Europäische Union mit der Schaffung der Währungsunion einen ersten, entscheidenden Schritt zur Anpassung ihrer Institutionen; sie war jedoch nicht in der Lage, sich auf eine echte Steuer- und Sozialpolitik für den Euro zu einigen. Später, mit dem Vertrag von Lissabon, stärkte sie die gesetzgebende Rolle des Europäischen Parlaments, scheiterte aber erneut daran, eine starke wirtschaftliche und politische Union zu schaffen, um den Euro zu vollenden. Infolgedessen war die EU nicht in der Lage, effektiv auf die ersten großen Herausforderungen und Krisen des 21. Jahrhunderts zu reagieren: den Finanzcrash von 2008, die Migrationsströme von 2015/2016, den Aufstieg des nationalen Populismus und das Brexit-Referendum im Jahr 2016. Dieses Versagen hat auch zu einer Stärkung der Rolle der nationalen Regierungen geführt – wie z.B. die mittlerweile übermäßige Machtkonzentration innerhalb des Europäischen Rates zeigt, dessen Handeln durch gegensätzliche nationale Vetos blockiert wird – und zu der chronischen Unfähigkeit der EU, eine gemeinsame Außenpolitik zu entwickeln, die in der Lage ist, die gemeinsamen strategischen Interessen Europas zu fördern. Nun aber hat sich die Tonlage geändert. Angesichts einer beispiellosen Krise der öffentlichen Gesundheit und des damit

Figure 9.6: Search for similar sentences in DATS

Bei der Konferenz zur Zuku

Score	Sentence	Type	Document name	Tags
0.99	Jetzt erwarten wir als europäische Bürgerinnen und Bürger mit Spannung den Beginn der Konferenz über die Zukunft Europas, einer Veranstaltung, die Bürgerinnen und Bürger, führende Vertreter der Zivilgesellschaft, Nichtregierungsorganisationen, Gewerkschaften und Vertreter nationaler und europäischer Institutionen zusammenbringen soll, um darüber zu debattieren und zu entscheiden, wie unsere Institutionen so angepasst werden können, dass der Aufbau unseres föderalen Europas vollendet wird.		84470 - Aufruf zur Konferenz zur Zukunft Europas getragen von über 400 Persönlichkeiten aus der ganzen Union (2021-05-20).txt	
0.99	DEVESA Vizepräsident der UEF und Sprecher der S&D im Ausschuss für konstitutionelle Fragen im Europäischen Parlament Gabriele BISCHOFF Vizepräsidentin der Europa-Union Deutschland (EUD), Vizepräsidentin im Ausschuss für konstitutionelle Fragen (AFCD) des Europäischen Parlaments Pascal DURAND Sprecher der Renew Europe Group im Ausschuss für konstitutionelle Fragen des Europäischen Parlaments Daniel FREUND Mitglied des Europäischen Parlaments (Grüne/EFA) und Vertreter im Executive Board der Konferenz zur Zukunft Europas Damian BOESELAGER Sprecher der		84470 - Aufruf zur Konferenz zur Zukunft Europas getragen von über 400 Persönlichkeiten aus der ganzen Union (2021-05-20).txt	

Figure 9.7: Similarity search result view in DATS

Document similarity search For a document similarity search, the following two options are planned: First, in the document search view one or more documents can be selected, and an additional icon in the multi-select action bar is added to start a document similarity search. Second, when viewing a single document, a new icon for document similarity search is added to the action bar. Both options trigger a document similarity search and the results are displayed in a list similar to the sentence search ordered by decreasing similarity.

Configurable dimensions-based similarity search For the dimensions-based similarity search, the UI would be extended in two aspects. First, the result view provides the combined similarity as well as the similarity per dimension and allows sorting by either score. Second, starting a dimensions-based similarity search via either any of the methods for sentence or document search first opens a dialog to configure the custom weighting of the different dimensions to specify how the combined similarity shall be computed.

9.3.2 Backend

Sentence similarity search The document preprocessing flow (see Figure 9.4) creates sentence and document embeddings, and stores them in a vector database using an approximate nearest neighbors (ANN) indices. To perform a similarity search, the API provides endpoints to directly query similar sentences given an existing sentence identifier or an arbitrary text. For the former, an optimized approach can be used: The embedding is already computed and the vector index is directly queried for similar sentences near the stored embedding with the sentence identifier. For the latter, the text is first sent to the ML container to obtain a sentence embedding representation. Then, the vector index is queried with the numeric vector representation.

Document similarity search The document similarity search is implemented analogously to the first case of sentence similarity. Instead of the sentence vector index, the document vector index is queried with the precomputed document embedding. There is no need for a just-in-time computation of the embedding, as a document-length query input is not provided. An alternative is asymmetric search where a relatively short text query is used to find a similar document. However, this requires an embedding model specialized for asymmetric semantic search, and is currently not integrated into DATS. In practice, full document similarity via fixed-length embeddings has issues because there is a limit on how well a single vector can represent an entire book *etc.* To mitigate this issue, DATS splits long documents into sub-documents as shown in the text preprocessing flow.

Configurable dimensions-based similarity search Supporting a dimensions-based similarity search with all functionalities is challenging for a production system that has to deal with large data sizes. When only searching for a single dimension, one vector index per dimension can be used for standard ANN retrieval with an additional decision beforehand which index to query. Searching for a mix of different dimensions is complex. A relatively performant option is to retrieve the k nearest neighbors for each dimension, look at the overlaps in the results, and filter and re-rank the results with the configured dimension weighting. However, it is perfectly possible that the combined result set is empty, especially when a too small value for k was used. In this case, the value of k would need to be increased until results are found. In

We refactored the model's source code and extracted the parts necessary for inference to put them into an installable Python package with a well-defined API called `maverick-coref-de` that is also made available on the Python package index¹⁶ and as a docker container¹⁷ for others to use. Finally, we added the `maverick-coref-de` library as a dependency to the platform's ML container, and created a minimal wrapper code to convert the input and output objects to be serializable and compatible with the Ray-based model server.

When the API container receives a request to perform coreference resolution on all unprocessed documents, it enqueues a job that is processed in the worker container. In that job all documents are queried from the database which have not yet been processed by the coreference resolution model. The documents are retrieved in batches, sent to the ML container to be processed efficiently as a batch, the returned coreference information is persisted in the database as annotations, and the document status is updated.

To store the coreference clusters as annotations, we altered the database schema by adding a *span group* entity. Annotations and span groups are connected as a many-to-many relationship with a link table to allow each span to be part of multiple groups, and each group to contain multiple spans. For analysis purposes, span groups enable an easy path to explore coreference clusters or other types of groups.

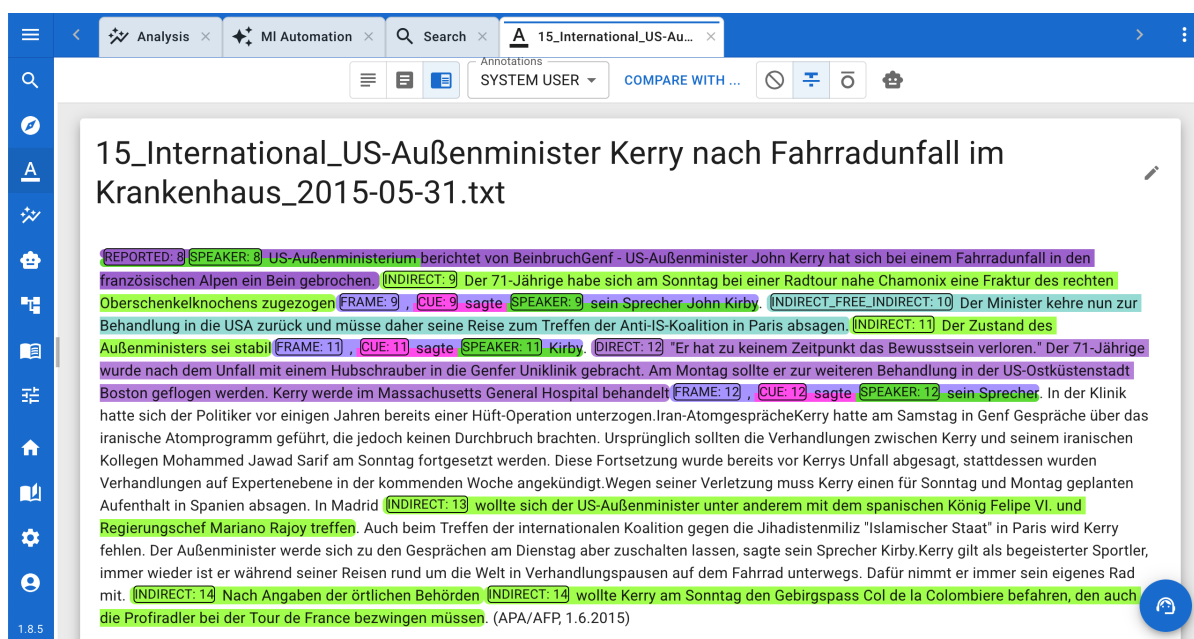


Figure 9.9: Quotation attribution visualization in DATS

9.5 Fine-grained Quotation Detection and Attribution

9.5.1 Frontend

The task of quotation detection and attribution relies on overlapping and nested annotations which can be displayed and edited in the annotation viewer (see Section 9.2.5). Figure 9.9 shows the rendering of a document with quotation attribution annotations. Compared to default ungrouped annotations, the quotation spans are part of a group that link the tuples such as speaker, cue and quote together. The group is displayed by complementing the code label with a numbered group identifier. As a single annotation span can be part of multiple groups, *i.e.* the same speaker annotation is connected to two instances of speech, a list of group identifiers is shown. Compared to the coreference annotations, each span still has a meaningful code label. Thus, the code's color is used for each annotation span regardless of its group identifier.

9.5.2 Backend

Similar to the coreference resolution model, the final quotation detection and attribution model *Quotect* from Chapter 5 can be integrated in the same ways. Again, we chose the integration as an on-demand job system task with the actual computation performed within the ML container.

The *Quotect* model is made available for easy inference with an installable Python package *quotect*¹⁸. The package is added as a dependency to the ML container, and a minimal wrapper for input and output objects integrate the library into the Ray-based model server.

The processing flow functions analogously to coreference resolution: Once the API is requested to perform quotation detection, it enqueues the corresponding job. The job is processed in a worker fetching unprocessed documents, sending these to the ML container in batches, and finally persisting the information in batches into the database. The database schema supports grouped annotations via *span groups* enabling a suitable persistent storage of quotation information obtained from the model within the relational database.

9.5.3 Combining Quotation and Coreference Information

As the full utility of the quotation detection and attribution model is only achieved together with a coreference resolution model, we explain how both models can be best connected within the platform. We will first discuss the desired UI before explaining the implementation in the backend.

Regarding the UI, there are two separate situations to consider. First, when viewing a single document with quotation annotations, the same speakers or addressees according to the coreference information shall be linked. The speaker/addressee could show the coreference chain identifier in addition to the quotation group identifier. Alternatively, each speaker/addressee instance could be augmented with the text of the longest member of the coreference chain,

¹⁶<https://pypi.org/project/maverick-coref-de/>

¹⁷see <https://github.com/uhh-lt/maverick-coref-de>

¹⁸<https://pypi.org/project/quotect>

typically enriching pronouns with the entities full name as specified in the text. Thereby, all quotations from the same speaker can be identified immediately.

Second, when looking at an aggregated view showing all quotations across a selected set of documents, speakers or addressees must use coreference information. Otherwise, the list mostly contains entries such as “he/she said ...”. Thus, speaker or addressee texts should always be replaced with the full name of the entity according to the information from each coreference chain. Thereby, quotations from many documents can be meaningfully searched by requiring the speaker text to contain a certain name *etc.* Further, quotations across documents can be grouped by the coreference-resolved entity text enabling straightforward count-based analysis to answer questions such as “Who was cited the most in the document collection ...”.

To enable the described UI functionalities, the speakers and addressees need to be linked to the best-suited speaker or addressee of the same coreference chain. Both outputs from the quotation and coreference model are stored as span annotations in the SQL database. The linking consists of three steps: Connecting coreference and quotations spans, selecting the most representative span per coreference chain, and finally linking each quotation speaker/addressee span to the chosen representative speaker/addressee span.

First, to combine both models’ information together, the best matching coreference mention span needs to be selected for each speaker or addressee span. This is achieved by comparing the token offsets of the spans and selecting the coreference span with the smallest token-based Levenshtein distance. This guarantees, the exact match is selected if available. Otherwise, the next best option is chosen which can occur *e.g.* if one span includes only a person’s full name while the other additionally contains the title *etc.*

Second, to choose the best representative span of a coreference chain, the longest token sequence with preference for earlier occurrence in the text can be used. However, as some coreference mentions contain long, descriptive information in relative clauses or parenthesis, such spans have to be filtered out by removing candidate spans with any commas or parenthesis.

Third, the selected coreference spans from step two are connected back to the quotation spans by choosing the best matching span analogously to step one. Then, the pairs of quotation spans belonging together can be linked by storing the resolved information in the database.

As the coreference resolution and quotation detection models can be run independently in any order in the job system, the combination of their information can be handled in another job that is scheduled after any of the other jobs finishes and checks if both required jobs have completed successfully for a document. Thus, even if a job is restarted due to an error or model change, the combination job is triggered to perform the linking process.

9.6 Annotation Scaling

9.6.1 Frontend

A good user experience is essential to semi-automatic functionalities such as the annotation scaling since a mediocre user experience can easily diminish the benefits of the automation, and pushes users to continue with the purely manual approach instead. Thus, the UI was designed with simplicity and speed in mind to optimally support researchers in their workflow. Annotation suggestions are shown when a user explicitly requests them by opening the

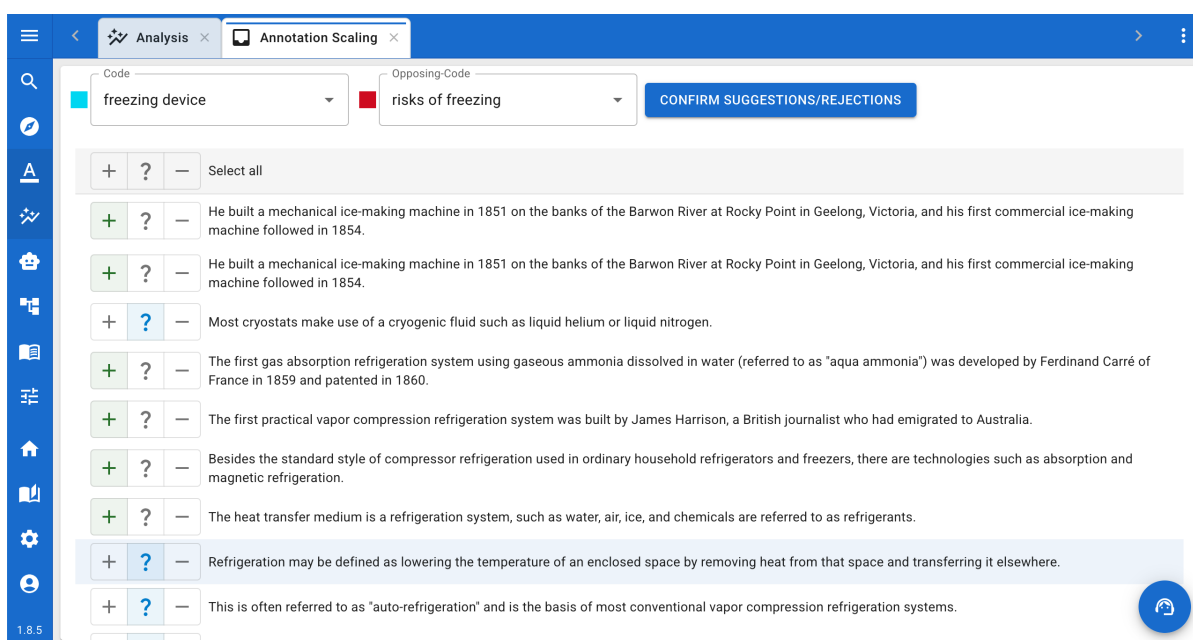


Figure 9.10: Semi-automatic annotation scaling in DATS

Annotation Scaling tab. This ensures that the user is not biased by the system, and it saves computation and energy. Users are free to annotate manually in documents opened in other tabs and mix/interleave suggestion-based annotation with manual coding.

Figure 9.10 shows the *Annotation Scaling* interface. The user selects a code to collect more annotations for and an opposing code to mark negatives as the annotation scaling method is based on binary classification. The system displays the first batch of suggestions. The suggestions are batch-approvable to label dozens of items with a single click or key press. Otherwise, suggestions can be individually labeled as correct or wrong. If no choice is made, the default is to ignore the suggestion without creating any code. Keyboard navigation with the arrow keys enables a fast labeling. Pressing the enter key or clicking the confirm button saves corresponding annotations and shows a newly computed batch of suggestions.

9.6.2 Backend

The integration of the interactive annotation scaling method from Chapter 7 in the production-grade platform supporting large amounts of data requires an elaborate implementation to (1) stay highly interactive with fast response times (2) without blocking the API for concurrent users, and (3) re-use existing data storage and processing models available.

The logic is implemented as a service running within the API container. Requests from the UI are handled synchronously to respond as quickly as possible. No long-running compute operations should be performed in the API container as this blocks the processing of request. Thus, all computations are delegated to other systems.

The first step to compute suggestions is to find the best-matching structure (*e.g.* sentence or token) to the annotated text segments of the user selected code. The SQL database is queried to compute the total distance in characters between all annotation spans and all available structure spans. The lowest, and thus best matching, structure is selected. In a second database

query, the identifier of the corresponding structure for each annotation for both codes is retrieved to (1) exclude already annotated passages, and (2) use the identifiers for the nearby query in the vector database.

The correct vector database (*e.g.* for sentences) is queried once with a batch of all identifiers to obtain internal object identifiers. These are used in second query to retrieve the top k most similar candidates per object. To filter the results, first all structures that are already annotated are removed. Then, in a third query the nearest object to each candidate is retrieved to check if a candidate is actually closest to the negative code. As this information is not stored in vector index, the SQL database is queried to get the annotation code of each object.

After accumulating all the necessary information from both databases, the final result list is constructed. The candidates are filtered again, the remaining candidates are sorted by their similarity to the initial similarity query with a positive code, duplicates are removed so that the first item is kept, and finally the top n candidates are returned as the suggestions.

Once the user confirms a batch of suggestions, normal annotations with the selected code are created in a batched operation in the SQL database. Afterwards, the process of finding new suggestions is started again with a now changed set of positive and negative examples leading to a new result set.

10 Conclusion

Abstract We conclude this dissertation by presenting and discussing the work shown in Parts II and III to answer the research questions raised during the introduction in Chapter 1 with the goal to bring state of the art (SotA) natural language processing (NLP) methods for large-scale text collections to the Digital Humanities (DH) and Computational Social Science (CSS). First, we summarize and discuss the newly developed NLP methods presented throughout Part II for application in DH and CSS. Then, we summarize the platform developed in Part III together with the integration of the NLP methods. We answer the research questions before ending with an outlook discussing envisioned extensions of the research platform and future directions for NLP methods in DH and CSS.

10.1	New NLP Methods for DH & CSS Workflows	179
10.1.1	Semantic Similarity Search for the Digital Humanities	179
10.1.2	Quotation Detection and Attribution	180
10.1.3	Transformer-based Coreference Resolution for German	180
10.1.4	Scaling Manual Annotations to a Large Corpus	181
10.2	Platform and NLP Method Application	181
10.2.1	Platform	182
10.2.2	Applying the NLP Methods in the Platform	182
10.3	Answering the Research Questions	183
10.4	Outlook	184
10.4.1	Envisioned Extensions of the Platform	184
10.4.2	Future Directions for NLP Methods in DH and CSS	185

10.1 New NLP Methods for DH & CSS Workflows

We present the NLP methods developed throughout Part II in a summarized form in this section and discuss their application in DH and CSS.

10.1.1 Semantic Similarity Search for the Digital Humanities

In Chapter 4, we presented two methods to enable semantic similarity search usage in interdisciplinary DH contexts: A configurable dimension-based similarity search and a semantic KWIC (Key Word in Context)-like search.

First, we introduced a novel semantic search configurable by the user to focus on different dimensions. Based on the SemEval 2022 Shared Task 8 dataset (Chen et al., 2022), we created multiple models focussing on the different similarity dimensions annotated in the dataset. The individually fine-tuned models perform well in comparison with human judgements. They achieve SotA performance on in-domain data for various dimensions and have successfully been applied to out-of-domain data. We used the models to construct a configurable semantic similarity search system and showed the applicability on experiments emulating the usage of such a system.

Our results indicate that the weighted combination of the different dimensions’ similarities is as suitable base to build systems providing a human-interpretable, configurable similarity search. Such a system enables a more interpretable notion of similarity by providing the similarity scores for different dimensions between documents. Further, it enables users to balance the aspects of similarity so that the search results better reflect what dimensions are important to the user for a specific query, thereby allowing answering queries such as “show me articles on the same topic but from different perspectives”.

Second, we presented an approach using structured similarity to find semantically similar units for variable-length spans. Our method uses linguistically informed structures to select span candidates and create embeddings. We showed that it is possible to retrieve variable-length entity spans using linguistic structures on top of contextualized word embeddings (CWEs) without any fine-tuning. Further, we found that only few labeled data points are needed for a k -nearest neighbors (k -NN) classification on the structured embeddings. Our method provides a semantic search for sub-sentence spans such as the widely used text-based KWIC method.

Overall, the work presented in Chapter 4 makes a large step towards a widely usable semantic similarity search for DH.

10.1.2 Quotation Detection and Attribution

In Chapter 5, we solved the problem of automatically detecting who said what to whom in German news articles by creating a new manually annotated dataset and developing our machine learning (ML) model *Quotect* to perform automatic quotation detection and attribution.

Our new dataset for quotation detection and attribution in German news articles is freely available under a Creative Commons license and provides curated, high-quality annotations. The fine-grained annotation schema allows the data to be used for various applications as it not only specifies who said what but also how, in which context, to whom, and it defines the type of quotation.

Quotect automatically identifies who said what to whom by performing fine-grained, high-quality quotation detection and attribution. The model predicts and connects five different types of quotations together with four different roles. The model is based on a sequence-to-sequence transformer architecture generating structured, linearized output from plain text using constrained decoding. We evaluated two linearization methods across multiple foundation models and model sizes in an ablation study, and performed a manual error analysis. Our models deliver a very strong performance on both a manual verification of the outputs and the evaluation metrics, almost doubling the scores of the two baseline systems.

Our *Quotect* system can be used for a range of use cases in DH, CSS and journalism. When combined with additional NLP methods such as coreference resolution or entity linking, identified quotations can easily be grouped and analyzed, thereby providing researchers and journalists new means to work with quotations in large document collections.

10.1.3 Transformer-based Coreference Resolution for German

In Chapter 6, we substantially improved the quality of automatic coreference resolution on German texts by adapting two coreference resolution model architectures developed for

English to German: The coarse-to-fine model and the Maverick model. On the established TüBa-D/Z dataset, we manage to achieve substantial improvements (up to +33.3 CoNLL F1 points) over the previous state of the art. On the SemEval-2010 dataset, our best models are comparable to the current state of the art while being highly efficient for applications on large-scale datasets; that is approximately 75 times faster and requiring approximately 350 times less computational resources. Our Maverick model performs strong on all available coreference resolution scoring metrics and in a manual evaluation on new data.

Overall, our coreference resolution models provide an efficient, robust way to automatically perform coreference resolution on German texts likely exceeding the annotation quality of any person who has not received extensive training or a strong background in linguistics. The automatically extracted coreference information greatly improves the utility of other NLP models such as quotation attribution, but also enables various analyses in DH and CSS contexts such as counting the occurrence frequency of entities of interest in a corpus.

10.1.4 Scaling Manual Annotations to a Large Corpus

In Chapter 7, we proposed and evaluated methods to semi-automatically scale manual annotations to large corpora by providing interactive suggestions from instantly adaptable classifiers and developed user interfaces (UIs) to make our methods usable for domain experts without programming or NLP skills.

The interactive method is based on CWEs with linguistic structures to create structured embeddings. In an initial one-time pre-processing phase, linguistic structure offsets are computed and stored in a database, and the structured embeddings are created from CWEs and stored in an approximate nearest neighbors (ANN) index. During interactive semi-automatic annotation scaling, the following steps are performed for annotations of a user-selected class: (1) Finding the best matching linguistic structure via minimal offset distance, (2) candidate retrieval by getting items near the selected structured embedding instances, (3) candidate filtering to remove already annotated spans or candidates closer to another class, and finally (4) the user accepts or discards suggestions to be stored in the database.

The quantitative annotation scaling is based on a k -NN classifier constructed from the annotations created manually or during the interactive step. All unannotated spans for the selected linguistic structures are classified according to the already annotated examples in the embedding-space proximity. As no training is performed, the results for the entire corpus are available within (milli)seconds.

Our evaluation shows that these methods can quickly scale annotations with minimal manual effort to large corpora to both obtain quantitative results and aggregated lists enabling a verification of the automated processing. Thus, we see our methods aiding qualitative researchers to bridge the gap to quantitative results and providing quantitative grounding for their hypotheses.

10.2 Platform and NLP Method Application

In this section, we summarize the platform developed in Part III together with the integration of the NLP methods.

10.2.1 Platform

In Chapter 8, we gathered platform requirements, specified design goals, and derived suitable system architectures fulfilling the design goals.

We discussed requirements for a DH and CSS research platform collected from experiences with project partners from CLICCS (Climate, Climate Change, and Society), D-WISE (Digitale Wissensoziologische Diskursanalyse), and from multiple collaborations at the Hub of Computing and Data Science (HCDS). The result is a list of important aspects that are lacking in existing software as discussed in Section 3.1: (1) Support of large datasets (both performance and automation functionality such as active learning or AI-in-the-loop), (2) elaborate search features (full-text and vector-based), (3) near real-time collaboration on the same data, (4) automatic entity network creation, (5) intuitive UI for DH and CSS researchers without programming experience, (6) exchangeable SotA NLP methods, and (7) an undogmatic design to be usable for many researchers and a wide variety of research questions. From the requirements we derived and discussed 10 design goals: Modularity, usability, configurability, widely applicable, interoperability, large-scale data performance, large-scale data methodology, interactivity, collaboration, and simple deployment.

Based on the design goals, we developed a number of architectures for the platform as a whole and its components. First, we define the overall architecture to be a client-server architecture with a web-based client and a server providing a unified, centralized API (Application Programming Interface) for the client. Second, we specify the client-server communication with a REST (REpresentational State Transfer)-based API, automatically generated on the basis of our OpenAPI specification. Third, we define the high-level server software architecture with three internal layers and external services such as a database. Then, we describe how the server can be scaled to process large data volumes and serve many concurrent users. Last, we define a suitable client architecture based on a single-page application (SPA).

Our developed platform architectures enable the creation of a robust software for production use in DH and CSS. All important requirements can be fulfilled while adhering to the design goals. With the presented platform we provide a theoretical foundation to implement a research platform for DH and CSS that is maintainable, and can greatly help researches in analyzing large quantities of data to answer their individual research questions using SotA NLP techniques, all without requiring programming knowledge.

10.2.2 Applying the NLP Methods in the Platform

In Chapter 9, we explained the actual implementation of the previously conceptualized platform as a collaborative effort in form of the discourse analysis tool suite (DATS), highlighted implementation aspects essential for the integration of the methods, and described the integration of the NLP methods from Part II into the platform with the required frontend UI and backend processing changes.

We specify the implementation of a set of essential components to integrate the previously developed NLP methods: (1) A powerful, flexible, connected database model providing the foundation to persist and retrieve information efficiently, (2) service interaction with proper interfaces to keep the backend maintainable, (3) deployment as multi-container application with orchestration, (4) text annotator for efficient manual annotation with support for overlapping spans, span groups, intelligent fast code selection, long documents support, while

retaining text readability, (5) database migrations to enable updating instances of the software while keeping any data created, (6) a robust, high-performance job system to handle large data sizes, properly utilize the available hardware, and allow flexible integration of new jobs.

We explained the integration of the developed NLP methods in the production-grade software platform with any required frontend UI and backend processing changes. The semantic similarity search is enabled by a mandatory pre-processing step using the job system to obtain various embeddings from a document and index these into a ANN for fast, interactive retrieval with the search UI. The annotation scaling method is integrated as a service within the API container and implements the interactive annotation suggestion method using multiple SQL (Structured Query Language) and ANN database queries to finally provide the results in a UI with a batched suggestion approval and keyboard support for an efficient workflow. Coreference resolution and quotation detection are both integrated as on-demand jobs with the job system using the ML container for the actual models' computations. The complex spans & span groups are supported in the data model, and the annotation visualizer is enhanced to show groups and color spans accordingly for coreference chains, and provide both annotation label and group identifier for quotation attribution tuples.

Our platform architecture and its implementation are well suited to integrate a wide range of NLP methods in a way that makes their usage highly interactive and straightforward even on large data collections. The research platform as a whole enables DH and CSS researches to analyze large text collections on their own terms, supported by SotA NLP techniques without requiring programming knowledge.

10.3 Answering the Research Questions

The overarching research question “How can state of the art natural language processing for large-scale text processing be brought to the Digital Humanities and Computational Social Science” is decomposed into the two research questions which will be answered in this section.

RQ1: Research Question 1

How can natural language processing methods be designed to be widely applicable to answer many Digital Humanities and Computational Social Science research questions?

Developing one or more specific NLP methods for each individual DH or CSS research question is infeasible as it does not scale to many use cases. Thus, the design of generic, widely-applicable NLP methods is required so that DH and CSS researchers can build upon the results to perform their specific analysis.

We developed several methods that can be used to help answer a wide range of DH and CSS research questions: A configurable dimension-based similarity search, a KWIC-like structured similarity search, quotation detection and attribution combined with coreference resolution, and semi-automatic scaling of manual annotations to large corpora. The methods are widely applicable because they are agnostic to the individual use case, can be used both in isolation and combination with other methods, and are configurable by the user for each specific scenario.

We found that NLP methods need to be designed with multiple use cases and reusability in mind as this helps a method to generalize to other use cases as well. This can further be

improved by providing configuration options for the NLP methods and allowing compositionality with other methods via simple pre- or post-processing. Overall, the NLP methods should not be designed to directly answer any DH or CSS research question. Instead, they should transform the data in such a way that a manual or statistical analysis can be performed on the new data to actually answer a specific research question.

RQ2: Research Question 2

How can a wide range of natural language processing methods be made available to Digital Humanities and Computational Social Science practitioners in interactive tooling?

NLP methods are usually not applicable in the void without additional tooling in DH or CSS contexts. Instead, NLP methods require an easy-to-use software platform providing all the necessary functionality to work on textual data in DH and CSS such as document import, automatic NLP-based information enrichment, collaboration features, (manual) annotation interface, high level of integration between all components, and a flexible undogmatic UI to support various use cases.

We found that a suitable software architecture is needed to integrate all kinds of NLP methods, both interactive and long-running tasks, with a unified data model for storing self-describing results, so UI components can be re-used as often as possible making the software easy-to-use as less specific behaviors need to be learned while also improving compositionality of the methods. Further, some NLP methods need to be adapted for production systems such as using an ANN index and using the specific data model for a deep integration to provide the most benefits.

Our developed platform provides a suitable foundation to integrate a wide range of novel NLP methods with support for large data quantities, and it features a UI that is easy-to-use for DH and CSS researchers. The integrated methods enable DH and CSS researchers to freely work with their data and analyze it to answer their specific research questions.

10.4 Outlook

Having summarized the key results and findings of this dissertation, we now present envisioned extensions of the platform that should provide significant benefits to its users, and we discuss possible future directions for NLP methods in DH and CSS.

10.4.1 Envisioned Extensions of the Platform

The connection between multiple ML models and a common UI via the linked data model is one of the core strengths of the implemented platform. This could be further improved so that the interoperation between models is highly flexible and seamless from a user perspective. Thus, users can dynamically combine the results from various models in their analyses. As an example, the annotation scaling method could additionally consider spans created from the quotation detection service allowing an easy scaling of speakers or quotations labeled with use case-specific codes. Any models should be made exchangeable and configurable on-the-fly so that users can select from a range of available models.

Quotation detection and attribution with coreference resolution is an excellent foundation to automatically create an advanced actor or entity network. The automatic generation of

such networks is a feature that was requested multiple times from various project partners during the time of this dissertation project. The quotation detection annotations can be used to filter entities by limiting them to those that participate in the discourse, and it allows more fine-grained interaction settings for network, *e.g.* requiring that two entities not only occur in the same document but rather that entity A speaks about entity B or vice versa. Coreference resolution is required for this to reach reasonable recall. Moreover, it allows to create an entity interaction network from all pairs of entities occurring in the same sentence — regardless whether they occur in their full form or only as a pronoun. This would greatly set the entity network apart from the few software systems integrating such a feature in a basic form.

Regarding the software architecture, a plugin system is the next logical step to integrate arbitrary models or services that might be specific to a platform instance or even a single project on said instance. Since there are no commonly agreed upon API standards for generic ML models, such a API should be specified for the platform so that any existing model or service can be integrated with a small wrapper. In the future, all models currently integrated into the platform should be refactored to be plugins as well, thereby making it even easier to change mandatory models such as text tokenization or language detection *etc.*

Once the platform is opened for external plugins, a generic yet customizable way to store and display results is required. The plugin API should also define a way to specify how results shall be stored in the existing database schema and how the UI should be constructed and filled with data for cases where the default views are insufficient.

Overall, the future direction of the platform is to support even more use cases and models on the basis of its solid yet flexible-enough foundation.

10.4.2 Future Directions for NLP Methods in DH and CSS

Generative large language models (LLMs) will likely be the method of choice for many DH and CSS researchers for the following reasons: (1) They are highly generic and can be used to perform almost any task, (2) they can be instructed via natural language without any coding knowledge, and (3) they can be integrated into research platforms to consume data, generate machine-readable outputs and potentially use other available tools.

However, not all DH and CSS researchers will adopt LLMs for their analysis for good reasons: (1) They are a black box with unknown internal workings, (2) they involve reproducibility issues especially with hosted proprietary models, (3) hallucinations in the output pose significant issues especially when results cannot be verified programmatically without resorting to a LLM for the verification task, (4) zero or few shot LLM performance is not up to par with specialized pre-trained language models (PLMs), and (5) LLMs require substantial computational resources with a noticeable environmental impact (*e.g.* by building or reactivating nuclear and fossil power plants to power LLM data centers).

As such, there two directions to pursue. First, methods and tools surrounding generative LLMs can be developed to mitigate the issues by *e.g.* providing provenance information, automatic linking of LLM outputs to the original input text, use of structured outputs instead of free-form text to improve automatic post-processing, more efficient locally hostable LLMs *etc.* When open-weight LLMs are integrated into a research platform that also provides other ways (not based on generative LLMs) to perform plausibility checks, the potential for negative impact of generative LLMs might be considerably reduced.

Second, NLP methods need to be developed and integrated into available software to become almost as easy-to-use as generative LLMs. This requires a wide range of NLP models, a UI that guides users to select the appropriate model for their task, and fully automatic visualization and verbalization of results.

NLP research in general follows the first option to some degree as explainable artificial intelligence (AI) has become an important aspect in the NLP community (Luo and Specia, 2024; Zhao et al., 2024; Resck et al., 2025). However, the leading industry corporations developing the current SotA generative models mainly push towards stronger, larger, and more complex models that shall be able to perform any task to eventually create an artificial general intelligence (AGI). Any alternative approaches will therefore always be compared to the ease of use and power of these proprietary LLM systems. Consequently, a collaborative effort is required for competitive alternatives to develop open, explainable approaches and make these methods available to DH and CSS communities via integration into user-friendly research platforms.

Bibliography

- Giuseppe Abrami, Manuel Stoeckel, and Alexander Mehler (2020). “TextAnnotator: A UIMA Based Tool for the Simultaneous and Collaborative Annotation of Texts”. eng. In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Ed. by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Marseille, France: European Language Resources Association, pp. 891–900. ISBN: 979-10-95546-34-4.
- Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf (2019). “FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Ed. by Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh. Minneapolis, Minnesota, USA: Association for Computational Linguistics, pp. 54–59. DOI: 10.18653/v1/N19-4010.
- Alan Akbik, Duncan Blythe, and Roland Vollgraf (2018). “Contextual String Embeddings for Sequence Labeling”. In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA, pp. 1638–1649. URL: <https://www.aclweb.org/anthology/C18-1139>.
- Alan Akbik and Martin Schenck (2013). “QuoteMine: A Repository of Newsworthy Quotes”. In: *Caleidoscope Abstract, German Society for Computational Linguistics and Language Technology (GSCL 2013)*. Ed. by Iryna Gurevych, Chris Biemann, and Torsten Zesch. Darmstadt, Germany.
- Mariana S. C. Almeida, Miguel B. Almeida, and André F. T. Martins (2014). “A Joint Model for Quotation Attribution and Coreference Resolution”. In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Ed. by Shuly Wintner, Sharon Goldwater, and Stefan Riezler. Gothenburg, Sweden: Association for Computational Linguistics, pp. 39–48. DOI: 10.3115/v1/E14-1005.
- Gene M. Amdahl (1967). “Validity of the single processor approach to achieving large scale computing capabilities”. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS ’67 (Spring). Atlantic City, New Jersey, USA: Association for Computing Machinery, pp. 483–485. ISBN: 9781450378956. DOI: 10.1145/1465482.1465560.
- Laurence Anthony (2024). *AntConc*. Center for English Language Education in Science and Engineering, School of Science and Engineering, Waseda University, Tokyo, Japan. URL: <https://www.laurenceanthony.net/software/antconc/releases/AntConc431/help.pdf> (visited on 09/04/2025).
- Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. arXiv: 1607.00450 [stat.ML].
- Jaeryang Baek, Ayana Hussain, Danny Liu, Nicholas Vincent, and Lawrence H. Kim (2025). *Open WebUI: An Open, Extensible, and Usable Interface for AI Interaction*. arXiv: 2510.02546 [cs.HC].
- Amit Bagga and Breck Baldwin (1998). “Entity-Based Cross-Document Coreferencing Using the Vector Space Model”. In: *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume*

1. Montreal, Quebec, Canada: Association for Computational Linguistics, pp. 79–85. DOI: 10.3115/980845.980859.
- David Bamman, Ted Underwood, and Noah A. Smith (2014). “A Bayesian Mixed Effects Model of Literary Character”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Kristina Toutanova and Hua Wu. Baltimore, Maryland, USA: Association for Computational Linguistics, pp. 370–379. DOI: 10.3115/v1/P14-1035.
- Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy (2024). “LLM2Vec: Large Language Models Are Secretly Powerful Text Encoders”. In: *First Conference on Language Modeling*. Philadelphia, Pennsylvania, USA. URL: <https://openreview.net/forum?id=IW1PR7vEBf>.
- Iz Beltagy, Matthew E. Peters, and Arman Cohan (2020). *Longformer: The Long-Document Transformer*. arXiv: 2004.05150 [cs.CL].
- Y. Bengio, P. Simard, and P. Frasconi (1994). “Learning Long-term Dependencies with Gradient Descent is Difficult”. In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181.
- Darina Benikova, Uli Fahrer, Alexander Gabriel, Manuel Kaufmann, Seid Muhie Yimam, Tatiana von Landesberger, and Chris Biemann (2014). “Network of the Day: Aggregating and Visualizing Entity Networks from Online Sources”. In: *Workshop proceedings of the 12th edition of the KONVENS conference*. Hildesheim, Germany, pp. 48–52.
- James Bergstra and Yoshua Bengio (2012). “Random Search for Hyper-parameter Optimization”. In: *Journal of Machine Learning Research (JMLR)* 13, pp. 281–305. ISSN: 1532-4435.
- David M. Berry, ed. (2012). *Understanding Digital Humanities*. Basingstoke, England: Palgrave Macmillan London. ISBN: 978-0-230-37193-4. DOI: doi.org/10.1057/9780230371934.
- Anders Björkelund and Jonas Kuhn (2014). “Learning Structured Perceptrons for Coreference Resolution with Latent Antecedents and Non-local Features”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland, USA: Association for Computational Linguistics, pp. 47–57. DOI: 10.3115/v1/P14-1005.
- David M. Blei (2012). “Probabilistic topic models”. In: *Commun. ACM* 55.4, pp. 77–84. ISSN: 0001-0782. DOI: 10.1145/2133806.2133826.
- David M. Blei, Andrew Y. Ng, and Michael I. Jordan (2003). “Latent dirichlet allocation”. In: *Journal of machine Learning research* 3, Jan, pp. 993–1022.
- Thomas Bögel and Michael Gertz (2015). “Did I Really Say That? – Combining Machine Learning and Dependency Relations to Extract Statements from German News Articles”. In: *Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology*. Duisburg-Essen, Germany: German Society for Computational Linguistics and Language Technology, pp. 13–21. URL: <https://konvens.org/proceedings/2015/GSCL-201504.pdf>.
- Bernd Bohnet, Chris Alberti, and Michael Collins (2023). “Coreference Resolution through a seq2seq Transition-Based System”. In: *Transactions of the Association for Computational Linguistics* 11, pp. 212–226. DOI: 10.1162/tacL_a_00543.
- Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov (2017). “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. DOI: 10.1162/tacL_a_00051.

- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 1877–1901.
- Annelen Brunner (2015). "Automatic recognition of speech, thought, and writing representation in German narrative texts". en. In: *Literary and Linguistic Computing* 28.4, pp. 563–575. ISSN: 1477-4615. DOI: 10.1093/llc/fqt024.
- Annelen Brunner, Stefan Engelberg, Fotis Jannidis, Ngoc Duyen Tanja Tu, and Lukas Weimer (2020a). "Corpus REDEWIEDERGABE". en. In: *Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC), May 11-16, 2020, Palais du Pharo, Marseille, France*. Ed. by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Paris, France: European Language Resources Association, pp. 803–812. ISBN: 979-10-95546-34-4.
- Annelen Brunner, Ngoc Duyen Tanja Tu, Lukas Weimer, and Fotis Jannidis (2019). "Deep learning for Free Indirect Representation". In: *Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019): Short Papers*. Erlangen, Germany: German Society for Computational Linguistics & Language Technology, pp. 241–245. URL: https://konvens.org/proceedings/2019/papers/KONVENS2019_paper_27.pdf.
- Annelen Brunner, Ngoc Duyen Tanja Tu, Lukas Weimer, and Fotis Jannidis (2020b). "To BERT or not to BERT - Comparing Contextual Embeddings in a Deep Learning Architecture for the Automatic Recognition of four Types of Speech, Thought and Writing Representation". In: *Proceedings of the 5th Swiss Text Analytics Conference and the 16th Conference on Natural Language Processing, SwissText/KONVENS 2020, Zurich, Switzerland, June 23-25, 2020 [online only]*. Ed. by Sarah Ebling, Don Tuggener, Manuela Hürlimann, Mark Cieliebak, and Martin Volk. Vol. 2624. CEUR Workshop Proceedings. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-2624/paper5.pdf>.
- Jiaxun Cai, Shexia He, Zuchao Li, and Hai Zhao (2018). "A Full End-to-End Semantic Role Labeler, Syntactic-agnostic Over Syntactic-aware?" In: *Proceedings of the 27th International Conference on Computational Linguistics*. Ed. by Emily M. Bender, Leon Derczynski, and Pierre Isabelle. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 2753–2765. URL: <https://aclanthology.org/C18-1233/>.
- Ben Carterette and Ellen M. Voorhees (2011). "Overview of Information Retrieval Evaluation". In: *Current Challenges in Patent Information Retrieval*. Ed. by Mihai Lupu, Katja Mayer, John Tait, and Anthony J. Trippe. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 69–85. ISBN: 978-3-642-19231-9. DOI: 10.1007/978-3-642-19231-9_3.
- Branden Chan, Stefan Schweter, and Timo Möller (2020). "German's Next Language Model". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Ed. by Donia

- Scott, Nuria Bel, and Chengqing Zong. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 6788–6796. DOI: 10.18653/v1/2020.coling-main.598.
- Xi Chen, Ali Zeynali, Chico Camargo, Fabian Flöck, Devin Gaffney, Przemyslaw Grabowicz, Scott A. Hale, David Jurgens, and Mattia Samory (2022). “SemEval-2022 Task 8: Multilingual news article similarity”. In: *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Ed. by Guy Emerson, Natalie Schluter, Gabriel Stanovsky, Ritesh Kumar, Alexis Palmer, Nathan Schneider, Siddharth Singh, and Shyam Ratan. Seattle, United States: Association for Computational Linguistics, pp. 1094–1106. DOI: 10.18653/v1/2022.semeval-1.155.
- Davide Chicco (2021). “Siamese Neural Networks: An Overview”. In: *Artificial Neural Networks*. Ed. by Hugh Cartwright. New York, New York, USA: Springer US, pp. 73–94. ISBN: 978-1-0716-0826-5. DOI: 10.1007/978-1-0716-0826-5_3.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.
- Krzysztof Marcin Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller (2021). “Rethinking Attention with Performers”. In: *International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=Ua6zuk0WRH>.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei (2022). *Scaling Instruction-Finetuned Language Models*. arXiv: 2210.11416 [cs.LG].
- Claudio Cioffi-Revilla (2010). “Computational social science”. In: *WIREs Computational Statistics* 2.3, pp. 259–271. DOI: 10.1002/wics.95.
- Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning (2019). “What Does BERT Look at? An Analysis of BERT’s Attention”. In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy, pp. 276–286. DOI: 10.18653/v1/W19-4828.
- Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning (2020). “ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators”. In: *8th International Conference on Learning Representations, ICLR 2020*. Addis Ababa, Ethiopia: OpenReview.net. URL: <https://openreview.net/forum?id=r1xMH1BtvB>.
- Kevin Clark and Christopher D. Manning (2016a). “Deep Reinforcement Learning for Mention-Ranking Coreference Models”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Ed. by Jian Su, Kevin Duh, and Xavier Carreras. Austin, Texas, USA: Association for Computational Linguistics, pp. 2256–2262. DOI: 10.18653/v1/D16-1245.

- Kevin Clark and Christopher D. Manning (2016b). “Improving Coreference Resolution by Learning Entity-Level Distributed Representations”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics, pp. 643–653. DOI: 10.18653/v1/P16-1061.
- David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan (1996). “Active learning with statistical models”. In: *J. Artif. Int. Res.* 4.1, pp. 129–145. ISSN: 1076-9757.
- Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov (2020). “Unsupervised Cross-lingual Representation Learning at Scale”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault. Online: Association for Computational Linguistics, pp. 8440–8451. DOI: 10.18653/v1/2020.acl-main.747.
- Juliet M. Corbin and Anselm C. Strauss (2015). *Basics of qualitative research*. en. 4th ed. Thousand Oaks, California, USA: SAGE Publications.
- John W. Creswell and J. David Creswell (2018). *Research design*. 5th ed. Thousand Oaks, California, USA: SAGE Publications. ISBN: 978-1-5063-8670-6.
- Scott A. Crossley, Laura K. Allen, Kristopher Kyle, and Danielle S. McNamara (2014). “Analyzing Discourse Processing Using a Simple Natural Language Processing Tool”. In: *Discourse Processes* 51.5-6, pp. 511–534. DOI: 10.1080/0163853X.2014.910723.
- Eugenio Culurciello (2018). *The fall of RNN / LSTM*. URL: <https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0> (visited on 10/22/2025).
- Karel D’Oosterlinck, Semere Kiros Bitew, Brandon Papineau, Christopher Potts, Thomas Demeester, and Chris Develder (2023). “CAW-coref: Conjunction-Aware Word-level Coreference Resolution”. In: *Proceedings of the Sixth Workshop on Computational Models of Reference, Anaphora and Coreference (CRAC 2023)*. Ed. by Maciej Ogrodniczuk, Vincent Ng, Sameer Pradhan, and Massimo Poesio. Singapore: Association for Computational Linguistics, pp. 8–14. DOI: 10.18653/v1/2023.crac-main.2.
- Max Dallabetta, Conrad Dobberstein, Adrian Breiding, and Alan Akbik (2024). *Zitatsuchmaschine*. URL: <https://zitatsuchmaschine.informatik.hu-berlin.de/> (visited on 01/21/2026).
- Tri Dao (2023). *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*. arXiv: 2307.08691 [cs.LG].
- Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré (2022). *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. arXiv: 2205.14135 [cs.LG].
- Angel Daza and Anette Frank (2018). “A Sequence-to-Sequence Model for Semantic Role Labeling”. In: *Proceedings of the Third Workshop on Representation Learning for NLP*. Ed. by Isabelle Augenstein, Kris Cao, He He, Felix Hill, Spandana Gella, Jamie Kiros, Hongyuan Mei, and Dipendra Misra. Melbourne, Australia: Association for Computational Linguistics, pp. 207–216. DOI: 10.18653/v1/W18-3027.
- Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni (2021). “Autoregressive Entity Retrieval”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: <https://openreview.net/forum?id=5k8F6UU39V>.

- Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman (1990). "Indexing by latent semantic analysis". In: *Journal of the American society for information science* 41.6, pp. 391–407.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota, USA: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.
- Marco Di Giovanni, Thomas Tasca, and Marco Brambilla (2022). "DataScience-Polimi at SemEval-2022 Task 8: Stacking Language Models to Predict News Article Similarity". In: *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Ed. by Guy Emerson, Natalie Schluter, Gabriel Stanovsky, Ritesh Kumar, Alexis Palmer, Nathan Schneider, Siddharth Singh, and Shyam Ratan. Seattle, United States: Association for Computational Linguistics, pp. 1229–1234. DOI: 10.18653/v1/2022.semeval-1.174.
- Vladimir Dobrovolskii (2021). "Word-Level Coreference Resolution". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 7670–7675. DOI: 10.18653/v1/2021.emnlp-main.605.
- Richard Eckart de Castilho, Éva Mújdricza-Maydt, Seid Muhie Yimam, Silvana Hartmann, Iryna Gurevych, Anette Frank, and Chris Biemann (2016). "A Web-based Tool for the Integrated Annotation of Semantic and Syntactic Structures". In: *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*. Osaka, Japan, pp. 76–84.
- Achim Edelmann, Tom Wolff, Danielle Montagne, and Christopher A. Bail (2020). "Computational Social Science and Sociology". In: *Annual Review of Sociology* 46. Volume 46, 2020, pp. 61–81. ISSN: 1545-2115. DOI: 10.1146/annurev-soc-121919-054621.
- Anton Ehrmanntraut, Julia Wunderle, Jan Pfister, Fotis Jannidis, and Andreas Hotho (2025). *ModernGBERT: German-only 1B Encoder Model Trained from Scratch*. arXiv: 2505.13136 [cs.CL].
- Jeffrey L. Elman (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.
- David Elson, Nicholas Dames, and Kathleen McKeown (2010). "Extracting Social Networks from Literary Fiction". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Ed. by Jan Hajič, Sandra Carberry, Stephen Clark, and Joakim Nivre. Uppsala, Sweden: Association for Computational Linguistics, pp. 138–147. URL: <https://aclanthology.org/P10-1015/>.
- Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Sibli, Dominik Krzemiński, Genta Indra Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Diganta Misra, Shreeya Dhakal, Jonathan Rystrom, Roman Solomatin, Ömer Veysel Çağatan, Akash Kundu, Martin Bernstorff, Shitao Xiao, Akshita Sukhlecha, Bhavish Pahwa, Rafał Poświata, Kranthi Kiran GV, Shawon Ashraf, Daniel Auras, Björn Plüster, Jan Philipp Harries, Loïc Magne, Isabelle Mohr, Dawei Zhu, Hippolyte Gisserot-Boukhlef, Tom Aarsen, Jan Kostkan, Konrad Wojtasik, Taemin Lee, Marek Suppa, Crystina Zhang, Roberta Rocca, Mohammed

- Hamdy, Andrianos Michail, John Yang, Manuel Faysse, Aleksei Vatolin, Nandan Thakur, Manan Dey, Dipam Vasani, Pranjal A. Chitale, Simone Tedeschi, Nguyen Tai, Artem Snegirev, Mariya Hendriksen, Michael Günther, Mengzhou Xia, Weijia Shi, Xing Han Lù, Jordan Clive, Gayatri K., Maksimova Anna, Silvan Wehrli, Maria Tikhonova, Henil Shalin Panchal, Aleksandr Abramov, Malte Ostendorff, Zheng Liu, Simon Clematide, Lester James Validad Miranda, Alena Fenogenova, Guangyu Song, Ruqiya Bin Safi, Wen-Ding Li, Alessia Borghini, Federico Cassano, Lasse Hansen, Sara Hooker, Chenghao Xiao, Vaibhav Adlakha, Orion Weller, Siva Reddy, and Niklas Muennighoff (2025). “MMTEB: Massive Multilingual Text Embedding Benchmark”. In: *The Thirteenth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=zl3pfz4VCV>.
- Xing Fang and Justin Zhan (2015). “Sentiment Analysis Using Product Review Data”. In: *Journal of Big Data* 2.1, p. 5. ISSN: 2196-1115. DOI: 10.1186/s40537-015-0015-2.
- Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang (2022). “Language-agnostic BERT Sentence Embedding”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, pp. 878–891. DOI: 10.18653/v1/2022.acl-long.62.
- Daniel Fernández-González (2023). “Transition-based semantic role labeling with pointer networks”. In: *Knowledge-Based Systems* 260, p. 110127. ISSN: 0950-7051. DOI: <https://doi.org/10.1016/j.knosys.2022.110127>.
- Roy Thomas Fielding (2000). “Architectural Styles and the Design of Network-based Software Architectures”. PhD thesis. Irvine, California, USA: University of California.
- Kathleen Fitzpatrick (2012). “The Humanities, Done Digitally”. In: *Debates in the Digital Humanities*. NED - New edition. University of Minnesota Press, pp. 12–15. ISBN: 9780816677948. (Visited on 02/09/2026).
- Jamie Folsom, Liam Andrew, Andrew Magliozzi, Daniel Collis-Puro, Seth Woodworth, Ayse Gursoy, Jacob Hilker, Ben Silverman, Peggy Wang, Lizao Wang, and Joshua Mbogo (2015). *Annotation Studio*. Web Application. URL: <https://www.annotationstudio.org> (visited on 12/03/2025).
- Yoav Freund and Robert E. Schapire (1999). “Large margin classification using the perceptron algorithm”. In: *Machine learning* 37.3, pp. 277–296.
- Alexander Fritzler, Varvara Logacheva, and Maksim KretoV (2019). “Few-shot classification in named entity recognition task”. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 993–1000.
- Phil Gage (1994). “A New Algorithm for Data Compression”. In: *The C User Journal*. URL: <http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM>.
- Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West (2023). “Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, pp. 10932–10952. DOI: 10.18653/v1/2023.emnlp-main.674.
- Felix A. Gers and Jürgen Schmidhuber (2000). “Recurrent nets that time and count”. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Vol. 3. Como, Italy: IEEE, pp. 189–194. ISBN: 0-7695-0619-4. DOI: 10.1109/ijcnn.2000.861302.

- Evelyn Gius, Jan Christoph Meister, Malte Meister, Marco Petris, Christian Bruck, Janina Jacke, Mareike Schumacher, Dominik Gerstorfer, Marie Flüh, and Jan Horstmann (2022). *CATMA: Computer Assisted Text Markup and Analysis*. Version 6.5.0. DOI: 10.5281/zenodo.6419805.
- Xavier Glorot and Yoshua Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterton. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. URL: <http://proceedings.mlr.press/v9/glorot10a.html>.
- Nikhil Goel and Ranjith Reddy Bommidi (2022). “Wolfies at SemEval-2022 Task 8: Feature extraction pipeline with transformers for Multi-lingual news article similarity”. In: *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Ed. by Guy Emerson, Natalie Schluter, Gabriel Stanovsky, Ritesh Kumar, Alexis Palmer, Nathan Schneider, Siddharth Singh, and Shyam Ratan. Seattle, United States: Association for Computational Linguistics, pp. 1129–1135. DOI: 10.18653/v1/2022.semeval-1.159.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press. ISBN: 9780262035613.
- Osamu Gotoh (1982). “An improved algorithm for matching biological sequences”. In: *Journal of Molecular Biology* 162.3, pp. 705–708. ISSN: 0022-2836.
- Maarten Grootendorst (2020). *KeyBERT: Minimal keyword extraction with BERT*. Version v0.3.0. DOI: 10.5281/zenodo.4461265.
- Talika Gupta, Hans Ole Hatzel, and Chris Biemann (2024). “Coreference in Long Documents using Hierarchical Entity Merging”. In: *Proceedings of the 8th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL 2024)*. Ed. by Yuri Bizzoni, Stefania Degaetano-Ortlieb, Anna Kazantseva, and Stan Szpakowicz. St. Julians, Malta: Association for Computational Linguistics, pp. 11–17. URL: <https://aclanthology.org/2024.latechclfl-1.2/>.
- Joseph Hajjar, Weicheng Ma, and Soroush Vosoughi (2022). “DartmouthCS at SemEval-2022 Task 8: Predicting Multilingual News Article Similarity with Meta-Information and Translation”. In: *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Ed. by Guy Emerson, Natalie Schluter, Gabriel Stanovsky, Ritesh Kumar, Alexis Palmer, Nathan Schneider, Siddharth Singh, and Shyam Ratan. Seattle, United States: Association for Computational Linguistics, pp. 1157–1162. DOI: 10.18653/v1/2022.semeval-1.163.
- Zellig S. Harris (1954). “Distributional structure”. In: *Word* 10.2-3, pp. 146–162.
- Trevor Hastie, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning*. 2nd ed. Springer Series in Statistics. New York, New York, USA: Springer. ISBN: 978-0-387-84858-7.
- Hua He, Denilson Barbosa, and Grzegorz Kondrak (2013). “Identification of Speakers in Novels”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Hinrich Schuetze, Pascale Fung, and Massimo Poesio. Sofia, Bulgaria: Association for Computational Linguistics, pp. 1312–1320. URL: <https://aclanthology.org/P13-1129/>.

- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90.
- Pengcheng He, Jianfeng Gao, and Weizhu Chen (2021). *DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing*. arXiv: 2111.09543 [cs.CL].
- Dan Hendrycks and Kevin Gimpel (2023). *Gaussian Error Linear Units (GELUs)*. arXiv: 1606.08415 [cs.LG].
- Erhard Hinrichs, Marie Hinrichs, and Thomas Zastrow (2010). “WebLicht: Web-Based LRT Services for German”. In: *Proceedings of the ACL 2010 System Demonstrations*. Ed. by Sandra Kübler. Uppsala, Sweden: Association for Computational Linguistics, pp. 25–29. URL: <https://aclanthology.org/P10-4005/>.
- Sepp Hochreiter and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.
- Andreas Holzinger (2016). “Interactive Machine Learning for Health Informatics: When Do We Need the Human-In-the-Loop?”. In: *Brain Informatics* 3.2, pp. 119–131.
- Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd (2020). *spaCy: Industrial-strength Natural Language Processing in Python*. <https://spacy.io/>.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White (1989). “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5, pp. 359–366. ISSN: 0893-6080. DOI: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8).
- Jan Horstmann (2019). *CATMA: Theory*. URL: <https://catma.de/philosophy/theory/> (visited on 01/22/2026).
- Yuxin Hou and Junming Huang (2025). “Natural language processing for social science research: A comprehensive review”. In: *Chinese Journal of Sociology* 11.1, pp. 121–157. DOI: 10.1177/2057150X241306780.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly (2019). “Parameter-efficient transfer learning for NLP”. In: *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. PMLR. Long Beach, California, USA, pp. 2790–2799.
- Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R. Bowman (2019). “Do Attention Heads in BERT Track Syntactic Dependencies?”. In: *Natural Language, Dialog and Speech (NDS) Symposium*. New York, New York, USA, pp. 1–7.
- Basra Jehangir, Saravanan Radhakrishnan, and Rahul Agarwal (2023). “A survey on Named Entity Recognition — datasets, tools, and methodologies”. In: *Natural Language Processing Journal* 3, p. 100017. ISSN: 2949-7191. DOI: <https://doi.org/10.1016/j.nlp.2023.100017>.
- J. Johnson, M. Douze, and H. Jegou (2021). “Billion-Scale Similarity Search with GPUs”. In: *IEEE Transactions on Big Data* 7.03, pp. 535–547. ISSN: 2332-7790. DOI: 10.1109/TBDATA.2019.2921572.
- Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy (2020). “SpanBERT: Improving Pre-training by Representing and Predicting Spans”. In: *Transactions of the Association for Computational Linguistics* 8. Ed. by Mark Johnson, Brian Roark, and Ani Nenkova, pp. 64–77. DOI: 10.1162/tac1_a_00300.

- Mandar Joshi, Omer Levy, Luke Zettlemoyer, and Daniel Weld (2019). “BERT for Coreference Resolution: Baselines and Analysis”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Hong Kong, China: Association for Computational Linguistics, pp. 5803–5808. doi: 10.18653/v1/D19-1588.
- Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov (2017). “Bag of Tricks for Efficient Text Classification”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 427–431. URL: <https://www.aclweb.org/anthology/E17-2068>.
- Daniel Jurafsky and James H. Martin (2026). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. 3rd. Online manuscript released January 6, 2026. URL: <https://web.stanford.edu/~jurafsky/slp3/>.
- Ben Kantor and Amir Globerson (2019). “Coreference Resolution with Entity Equalization”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, pp. 673–677. doi: 10.18653/v1/P19-1066.
- Andrej Karpathy (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: <http://karpathy.github.io/2015/05/21/rnn-effectiveness> (visited on 10/22/2025).
- Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret (2020). *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*. arXiv: 2006.16236 [cs.LG].
- Eliyahu Kiperwasser and Yoav Goldberg (2016). “Simple and accurate dependency parsing using bidirectional LSTM feature representations”. In: *Transactions of the Association for Computational Linguistics* 4, pp. 313–327.
- Yuval Kirstain, Ori Ram, and Omer Levy (2021). “Coreference Resolution without Span Representations”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Ed. by Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli. Online: Association for Computational Linguistics, pp. 14–19. doi: 10.18653/v1/2021.acl-short.3.
- Manfred Klenner and Don Tuggener (2011). “An Incremental Entity-Mention Model for Coreference Resolution with Restrictive Antecedent Accessibility”. In: *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*. Hissar, Bulgaria: Association for Computational Linguistics, pp. 178–185.
- Jan-Christoph Klie, Michael Bugert, Beto Boullosa, Richard Eckart de Castilho, and Iryna Gurevych (2018). “The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation”. In: *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*. Ed. by Dongyan Zhao. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 5–9. URL: <https://aclanthology.org/C18-2002/>.
- Thomas Kluyver, Benjamin Ragan-kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain

- Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing (2016). “Jupyter Notebooks—a publishing format for reproducible computational workflows”. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87–90. ISSN: 0015-0193. DOI: 10.3233/978-1-61499-649-1-87.
- Gertraud Koch, Chris Biemann, Isabel Eiser, Tim Fischer, Florian Schneider, Teresa Stumpf, and Alejandra Tijerina García (2022). “D-WISE Tool Suite for the Sociology of Knowledge Approach to Discourse”. In: *Culture and Computing*. Ed. by Matthias Rauterberg. Cham: Springer International Publishing, pp. 68–83. ISBN: 978-3-031-05434-1.
- Martin Köppelmann, Dustin Lange, Claudia Lehmann, Marika Marszalkowski, Felix Naumann, Peter Retzlaff, Sebastian Stange, and Voget Lea (2012). *Scalable Similarity Search with Dynamic Similarity Measures*. Tech. rep. Hasso Plattner Institute, Potsdam, Germany.
- Ralf Krestel, Sabine Bergler, and René Witte (2008). “Minding the Source: Automatic Tagging of Reported Speech in Newspaper Articles”. In: *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC’08)*. Ed. by Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, and Daniel Tapias. Marrakech, Morocco: European Language Resources Association (ELRA). URL: <https://aclanthology.org/L08-1249/>.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. Lake Tahoe, Nevada, USA, pp. 1097–1105.
- Markus Krug, Frank Puppe, Fotis Jannidis, Luisa Macharowsky, Isabella Reger, and Lukas Weimar (2015). “Rule-based Coreference Resolution in German Historic Novels”. In: *Proceedings of the Fourth Workshop on Computational Linguistics for Literature*. Ed. by Anna Feldman, Anna Kazantseva, Stan Szpakowicz, and Corina Koolen. Denver, Colorado, USA: Association for Computational Linguistics, pp. 98–104. DOI: 10.3115/v1/W15-0711.
- Markus Krug, Lukas Weimer, Isabella Reger, Luisa Macharowsky, Stephan Feldhaus, Frank Puppe, and Fotis Jannidis (2018). “Description of a Corpus of Character References in German Novels – DROC [Deutsches Roman Corpus]”. In: *DARIAH-DE Working Papers*. 27. Göttingen: DARIAH-DE. URL: <http://webdoc.sub.gwdg.de/pub/mon/dariah-de/dwp-2018-27.pdf>.
- Veronika Laippala, Samuel Rönqvist, Miika Oinonen, Aki-Juhani Kyröläinen, Anna Salmela, Douglas Biber, Jesse Egbert, and Sampo Pyysalo (2022). “Register identification from the unrestricted open Web using the Corpus of Online Registers of English”. In: *Language Resources and Evaluation*, pp. 1–35. DOI: 10.1007/s10579-022-09624-1.
- David M. J. Lazer, Alex Pentland, Duncan J. Watts, Sinan Aral, Susan Athey, Noshir Contractor, Deen Freelon, Sandra Gonzalez-Bailon, Gary King, Helen Margetts, Alondra Nelson, Matthew J. Salganik, Markus Strohmaier, Alessandro Vespignani, and Claudia Wagner (2020). “Computational social science: Obstacles and opportunities”. In: *Science* 369.6507, pp. 1060–1062. DOI: 10.1126/science.aaz8170.
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *Nature* 521, pp. 436–444. DOI: 10.1038/nature14539.
- Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky (2011). “Stanford’s Multi-Pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task”. In: *Proceedings of the Fifteenth Conference on Computational Natural*

- Language Learning: Shared Task*. Ed. by Sameer Pradhan. Portland, Oregon, USA: Association for Computational Linguistics, pp. 28–34. URL: <https://aclanthology.org/W11-1902/>.
- Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer (2017). “End-to-end Neural Coreference Resolution”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Copenhagen, Denmark: Association for Computational Linguistics, pp. 188–197. DOI: 10.18653/v1/D17-1018.
- Kenton Lee, Luheng He, and Luke Zettlemoyer (2018). “Higher-Order Coreference Resolution with Coarse-to-Fine Inference”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana, USA: Association for Computational Linguistics, pp. 687–692. DOI: 10.18653/v1/N18-2108.
- David D. Lewis and William A. Gale (1994). “A Sequential Algorithm for Training Text Classifiers”. In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Dublin, Ireland, pp. 3–12.
- Xiangang Li and Xihong Wu (2015). “Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. Brisbane, Australia, pp. 4520–4524.
- Xiaoya Li, Xiaofei Sun, Yuxian Meng, Junjun Liang, Fei Wu, and Jiwei Li (2020a). “Dice Loss for Data-imbalanced NLP Tasks”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault. Online: Association for Computational Linguistics, pp. 465–476. DOI: 10.18653/v1/2020.acl-main.45.
- Zuchao Li, Shexia He, Hai Zhao, Yiqing Zhang, Zhuosheng Zhang, Xi Zhou, and Xiang Zhou (2019). “Dependency or Span, End-to-End Uniform Semantic Role Labeling”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01, pp. 6730–6737. DOI: 10.1609/aaai.v33i01.33016730.
- Zuchao Li, Hai Zhao, Rui Wang, and Kevin Parnow (2020b). “High-order Semantic Role Labeling”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Ed. by Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, pp. 1134–1151. DOI: 10.18653/v1/2020.findings-emnlp.102.
- Tianyu Liu, Yuchen Eleanor Jiang, Nicholas Monath, Ryan Cotterell, and Mrinmaya Sachan (2022). “Autoregressive Structured Prediction with Language Models”. In: *Findings of the Association for Computational Linguistics: EMNLP 2022*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, pp. 993–1005. DOI: 10.18653/v1/2022.findings-emnlp.70.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv: 1907.11692 [cs.CL].
- Ilya Loshchilov and Frank Hutter (2019). “Decoupled Weight Decay Regularization”. In: *7th International Conference on Learning Representations, ICLR 2019*. New Orleans, Louisiana, USA: OpenReview.net. URL: <https://openreview.net/forum?id=Bkg6RiCqY7>.

- Haoyan Luo and Lucia Specia (2024). *From Understanding to Utilization: A Survey on Explainability for Large Language Models*. arXiv: 2401.12874 [cs.CL].
- Xiaoqiang Luo (2005). “On Coreference Resolution Performance Metrics”. In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Ed. by Raymond Mooney, Chris Brew, Lee-Feng Chien, and Katrin Kirchhoff. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 25–32. URL: <https://aclanthology.org/H05-1004/>.
- Ruotian Ma, Xin Zhou, Tao Gui, Yiding Tan, Linyang Li, Qi Zhang, and Xuanjing Huang (2022). “Template-free Prompt Tuning for Few-shot NER”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, Washington, USA, pp. 5721–5732.
- Yu A. Malkov and Dmitry A. Yashunin (2018). “Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4, pp. 824–836.
- Markus Mandalka (2023). *Open Semantic Search*. Web Application. URL: <https://opensemanticticsearch.org> (visited on 02/19/2026).
- Christopher D. Manning and Hinrich Schütze (1999). *Foundations of statistical natural language processing*. Cambridge, Massachusetts, USA: MIT Press. ISBN: 0262133601.
- Giuliano Martinelli, Edoardo Barba, and Roberto Navigli (2024). “Maverick: Efficient and Accurate Coreference Resolution Defying Recent Trends”. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, pp. 13380–13394. DOI: 10.18653/v1/2024.acl-long.722.
- Andrew Kachites McCallum (2002). *MALLET: A Machine Learning for Language Toolkit*. Online. URL: <http://mallet.cs.umass.edu> (visited on 02/19/2026).
- Barbara McGillivray, Thierry Poibeau, and Pablo Ruiz (2020). “Digital Humanities and Natural Language Processing: “Je t’aime... Moi non plus””. In: *Digital Humanities Quarterly* 14.2. DOI: 10.17863/CAM.55816.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations (ICLR), Workshop Track Proceedings*. Scottsdale, Arizona, USA.
- Bonan Min, Hayley Ross, Elinor Sulem, Amir Poursan Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth (2023). “Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey”. In: *ACM Comput. Surv.* 56.2. ISSN: 0360-0300. DOI: 10.1145/3605943.
- Marvin Minsky (1987). *Perceptrons: An Introduction to Computational Geometry, Expanded Edition*. The MIT Press. ISBN: 0262631113.
- Nafise Sadat Moosavi and Michael Strube (2016). “Which Coreference Evaluation Metric Do You Trust? A Proposal for a Link-based Entity Aware Metric”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics, pp. 632–642. DOI: 10.18653/v1/P16-1060.
- Giovanni Moretti, Sara Tonelli, Stefano Menini, and R. Sprugnoli (2014). “ALCIDE: An online platform for the Analysis of Language and Content In a Digital Environment”. English. In: *Proceedings of the First Italian Conference on Computational Linguistics CLiC-it 2014 & the*

- Technology in Society* 66, p. 101647. ISSN: 0160-791X. DOI: <https://doi.org/10.1016/j.techsoc.2021.101647>.
- Shon Otmazgin, Arie Cattan, and Yoav Goldberg (2023). “LingMess: Linguistically Informed Multi Expert Scorers for Coreference Resolution”. In: *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. Ed. by Andreas Vlachos and Isabelle Augenstein. Dubrovnik, Croatia: Association for Computational Linguistics, pp. 2752–2760. DOI: 10.18653/v1/2023.eacl-main.202.
- Julio-Omar Palacio-Niño and Fernando Berzal (2019). *Evaluation Metrics for Unsupervised Learning Algorithms*. arXiv: 1905.05667 [cs.LG].
- Giovanni Paolini, Ben Athiwaratkun, Jason Krone, Jie Ma, Alessandro Achille, RISHITA ANUBHAI, Cicero Nogueira dos Santos, Bing Xiang, and Stefano Soatto (2021). “Structured Prediction as Translation between Augmented Natural Languages”. In: *International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=US-TP-xnXI>.
- Sean Papay and Sebastian Padó (2019). “Quotation Detection and Classification with a Corpus-Agnostic Model”. In: *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*. Ed. by Ruslan Mitkov and Galia Angelova. Varna, Bulgaria: INCOMA Ltd., pp. 888–894. DOI: 10.26615/978-954-452-056-4_103.
- Silvia Pareti (2012). “A Database of Attribution Relations”. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC’12)*. Ed. by Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Istanbul, Turkey: European Language Resources Association (ELRA), pp. 3213–3217. URL: <https://aclanthology.org/L12-1571/>.
- Silvia Pareti, Tim O’Keefe, Ioannis Konstas, James R. Curran, and Irena Koprinska (2013). “Automatically Detecting and Attributing Indirect Quotations”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Ed. by David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard. Seattle, Washington, USA: Association for Computational Linguistics, pp. 989–999. URL: <https://aclanthology.org/D13-1101/>.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio (2013). “On the Difficulty of Training Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning*. Vol. 28. ICML’13. Atlanta, Georgia, USA: JMLR.org, pp. III-1310–III-1318.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). “Deep Contextualized Word Representations”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by

- Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana, USA: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.
- Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych (2021). “AdapterFusion: Non-Destructive Task Composition for Transfer Learning”. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Ed. by Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty. Online: Association for Computational Linguistics, pp. 487–503. DOI: 10.18653/v1/2021.eacl-main.39.
- Jan Pfister, Julia Wunderle, and Andreas Hotho (2024). *LLäMmlein: Compact and Competitive German-Only Language Models from Scratch*. arXiv: 2411.11171 [cs.CL].
- Ian Porada, Xiyuan Zou, and Jackie Chi Kit Cheung (2024). “A Controlled Reevaluation of Coreference Resolution Models”. In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. Torino, Italia: ELRA and ICCL, pp. 256–263. URL: <https://aclanthology.org/2024.lrec-main.23/>.
- Horst Pöttker (2003). “News and its communicative quality: the inverted pyramid—when and why did it appear?”. In: *Journalism Studies* 4.4, pp. 501–511. DOI: 10/fwvrf4.
- Bruno Pouliquen, Ralf Steinberger, and Clive Best (2007). “Automatic Detection of Quotations in Multilingual News”. In: *The International Conference on Recent Advances in Natural Language Processing*. RANLP 2007. Borovets, Bulgaria, pp. 487–492. URL: <http://lml.bas.bg/ranlp2007/DOCS/RANLP2007.pdf>.
- Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong (2013). “Towards Robust Linguistic Analysis using OntoNotes”. In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Ed. by Julia Hockenmaier and Sebastian Riedel. Sofia, Bulgaria: Association for Computational Linguistics, pp. 143–152. URL: <https://aclanthology.org/W13-3516/>.
- Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang (2012). “CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes”. In: *Joint Conference on EMNLP and CoNLL - Shared Task*. Ed. by Sameer Pradhan, Alessandro Moschitti, and Nianwen Xue. Jeju Island, Korea: Association for Computational Linguistics, pp. 1–40. URL: <https://aclanthology.org/W12-4501/>.
- James Pustejovsky and Amber Stubbs (2012). *Natural Language Annotation for Machine Learning - a Guide to Corpus-Building for Applications*. O’Reilly. ISBN: 978-1-449-30666-3.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018). *Improving Language Understanding by Generative Pre-Training*. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf. (Visited on 01/22/2026).
- Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). “Language Models are Unsupervised Multitask Learners”. In: URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (visited on 01/22/2026).

- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu (2020). “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *J. Mach. Learn. Res.* 21.1. ISSN: 1532-4435.
- Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning (2010). “A Multi-Pass Sieve for Coreference Resolution”. In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Ed. by Hang Li and Lluís Màrquez. Cambridge, Massachusetts, USA: Association for Computational Linguistics, pp. 492–501. URL: <https://aclanthology.org/D10-1048/>.
- Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He (2020). “ZeRO: memory optimizations toward training trillion parameter models”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’20. Atlanta, Georgia, USA: IEEE Press. ISBN: 9781728199986.
- Prajit Ramachandran, Barret Zoph, and Quoc V. Le (2017). *Searching for Activation Functions*. arXiv: 1710.05941 [cs.NE].
- Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He (2020). “DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’20. Virtual Event, California, USA: Association for Computing Machinery, pp. 3505–3506. ISBN: 9781450379984. DOI: 10.1145/3394486.3406703.
- Marta Recasens, Lluís Màrquez, Emili Sapena, M. Antònia Martí, Mariona Taulé, Véronique Hoste, Massimo Poesio, and Yannick Versley (2010). “SemEval-2010 Task 1: Coreference Resolution in Multiple Languages”. In: *Proceedings of the 5th International Workshop on Semantic Evaluation*. Ed. by Katrin Erk and Carlo Strapparava. Uppsala, Sweden: Association for Computational Linguistics, pp. 1–8. URL: <https://aclanthology.org/S10-1001/>.
- Nils Reimers and Iryna Gurevych (2019). “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Hong Kong, China: Association for Computational Linguistics, pp. 3982–3992. DOI: 10.18653/v1/D19-1410.
- Lucas Resck, Isabelle Augenstein, and Anna Korhonen (2025). “Explainability and Interpretability of Multilingual Large Language Models: A Survey”. In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng. Suzhou, China: Association for Computational Linguistics, pp. 20454–20486. ISBN: 979-8-89176-332-6. DOI: 10.18653/v1/2025.emnlp-main.1033.
- Stephen Robertson and Hugo Zaragoza (2009). “The Probabilistic Relevance Framework: BM25 and Beyond”. In: *Found. Trends Inf. Retr.* 3.4, pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/15000000019.
- Ina Roesiger and Jonas Kuhn (2016). “IMS HotCoref DE: A Data-Driven Co-Reference Resolver for German”. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*. Portorož, Slovenia: European Language Resources Association (ELRA), pp. 155–160.

- Frank Rosenblatt (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.
- Jan Oliver Rüdiger (2018). *CorpusExplorer*. Download. Universität Kassel – Universität Siegen. URL: <http://corpusexplorer.de> (visited on 01/22/2026).
- Andrew Rueda, Elena Alvarez-Mellado, and Constantine Lignos (2024). “CoNLL#: Fine-grained Error Analysis and a Corrected Test Set for CoNLL-03 English”. In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. Torino, Italia: ELRA and ICCL, pp. 3718–3728. URL: <https://aclanthology.org/2024.lrec-main.330/>.
- David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams (1986). “Learning representations by back-propagating errors”. In: *Nature* 323, pp. 533–536. DOI: 10.1038/323533a0.
- Eugen Ruppert, Dirk Hartung, Phillip Sittig, Tjorben Gschwander, Lennart Rönneburg, Tobias Killing, and Chris Biemann (2018). “LawStats – Large-Scale German Court Decision Evaluation Using Web Service Classifiers”. In: *Machine Learning and Knowledge Extraction: Second IFIP TC 5, TC 8/WG 8.4, 8.9, TC 12/WG 12.9 International Cross-Domain Conference, CD-MAKE 2018, Hamburg, Germany, August 27–30, 2018, Proceedings*. Hamburg, Germany: Springer-Verlag, pp. 212–222. ISBN: 978-3-319-99739-1. DOI: 10.1007/978-3-319-99740-7_14.
- Christian Scheible, Roman Klinger, and Sebastian Padó (2016). “Model Architectures for Quotation Detection”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics, pp. 1736–1745. DOI: 10.18653/v1/P16-1164.
- Timo Schick and Hinrich Schütze (2021). “Exploiting Cloze-Questions for Few-Shot Text Classification and Natural Language Inference”. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Ed. by Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty. Online: Association for Computational Linguistics, pp. 255–269. DOI: 10.18653/v1/2021.eacl-main.20.
- Jürgen Schmidhuber (2015). “Deep learning in neural networks: An overview”. In: *Neural networks* 61, pp. 85–117.
- Susan Schreibman, Siemens Ray, and John Unsworth, eds. (2004). *A Companion to Digital Humanities*. Blackwell. ISBN: 9780470999875. DOI: 10.1002/9780470999875.
- Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao (2024). *FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision*. arXiv: 2407.08608 [cs.LG].
- Eyal Shnarch, Alon Halfon, Ariel Gera, Marina Danilevsky, Yannis Katsis, Leshem Choshen, Martin Santillan Cooper, Dina Epelboim, Zheng Zhang, Dakuo Wang, Lucy Yip, Liat Ein-Dor, Lena Dankin, Ilya Shnayderman, Ranit Aharonov, Yunyao Li, Naftali Liberman, Philip Levin Slesarev, Gwilym Newton, Shila Ofek-Koifman, Noam Slonim, and Yoav Katz (2022). “Label Sleuth: From Unlabeled Text to a Classifier in a Few Hours”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.
- Rainer Simon, Elton Barker, Leif Isaksen, and Pau de Soto Cañamares (2017). “Linked data annotation without the pointy brackets: Introducing Recogito 2”. In: *Journal of Map & Geography Libraries* 13.1, pp. 111–132.

- Stéfan Sinclair and Geoffrey Rockwell (2016). *Voyant Tools*. Web Application. URL: <https://voyant-tools.org> (visited on 01/22/2026).
- Iknoor Singh, Yue Li, Melissa Thong, and Carolina Scarton (2022). “GateNLP-UShef at SemEval-2022 Task 8: Entity-Enriched Siamese Transformer for Multilingual News Article Similarity”. In: *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Ed. by Guy Emerson, Natalie Schluter, Gabriel Stanovsky, Ritesh Kumar, Alexis Palmer, Nathan Schneider, Siddharth Singh, and Shyam Ratan. Seattle, United States: Association for Computational Linguistics, pp. 1121–1128. DOI: 10.18653/v1/2022.semeval-1.158.
- Jake Snell, Kevin Swersky, and Richard Zemel (2017). “Prototypical Networks for Few-shot Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30.
- Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski (2019). “Matching the Blanks: Distributional Similarity for Relation Learning”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David R. Traum, and Lluís Màrquez. Florence, Italy, pp. 2895–2905. DOI: 10.18653/v1/p19-1279.
- Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). “Dropout: a simple way to prevent neural networks from overfitting”. In: *Journal of Machine Learning Research (JMLR)* 15.1, pp. 1929–1958.
- Josef Steinberger, Massimo Poesio, Mijail A. Kabadjov, and Karel Ježek (2007). “Two uses of anaphora resolution in summarization”. In: *Information Processing & Management* 43.6, pp. 1663–1680.
- Pontus Stenetorp, 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*. Ed. by Frédérique Segond. Avignon, France: Association for Computational Linguistics, pp. 102–107. URL: <https://aclanthology.org/E12-2021/>.
- Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum (2018). “Linguistically-Informed Self-Attention for Semantic Role Labeling”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium, pp. 5027–5038. DOI: 10.18653/v1/D18-1548.
- Omri Suissa, Avshalom Elmalech, and Maayan Zhitomirsky-Geffet (2022). “Text analysis using deep neural networks in digital humanities and information science”. In: *Journal of the Association for Information Science and Technology* 73.2, pp. 268–287. DOI: <https://doi.org/10.1002/asi.24544>.
- Ilya Sutskever, Oriol Vinyals, and Quoc V. Le (2014). “Sequence to Sequence Learning with Neural Networks”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. NIPS’14*. Montreal, Canada: MIT Press, pp. 3104–3112.
- Ben Taskar, Pieter Abbeel, and Daphne Koller (2002). “Discriminative probabilistic models for relational data”. In: *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence. UAI’02*. Alberta, Canada: Morgan Kaufmann Publishers Inc., pp. 485–492. ISBN: 1558608974.
- Deep Search Team (2024). *Docling Technical Report*. Tech. rep. Version 1.0.0. DOI: 10.48550/arXiv.2408.09869. eprint: 2408.09869.

- Gemma Team et al. (2025). *Gemma 3 Technical Report*. arXiv: 2503.19786 [cs.CL].
- Heike Telljohann, Erhard Hinrichs, and Sandra Kübler (2004). “The Tüba-D/Z Treebank: Annotating German with a Context-Free Backbone”. In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC’04)*. Ed. by Maria Teresa Lino, Maria Francisca Xavier, Fátima Ferreira, Rute Costa, and Raquel Silva. Lisbon, Portugal: European Language Resources Association (ELRA). URL: <https://aclanthology.org/L04-1096/>.
- Heike Telljohann, Erhard W. Hinrichs, Sandra Kübler, Heike Zinsmeister, and Kathrin Beck (2017). “Stylebook for the Tübingen treebank of written German (TüBa-D/Z)”. In: *Seminar für Sprachwissenschaft, Universität Tübingen, Germany*.
- Erik F. Tjong Kim Sang 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*, pp. 142–147. URL: <https://aclanthology.org/W03-0419/>.
- Maxim Tkachenko, Mikhail Malyuk, Andrey Holmanyuk, and Nikolai Liubimov (2020). *Label Studio: Data labeling software*. URL: <https://github.com/HumanSignal/label-studio> (visited on 01/22/2026).
- Maurizio Toscano, Manuel Jesús Cobo Martín, and Enrique Herrera Viedma (2022). *Software solutions for web information systems in digital humanities: review, analysis and comparative study*. DOI: 10.3145/epi.2022.mar.11.
- Shubham Toshniwal, Sam Wiseman, Allyson Ettinger, Karen Livescu, and Kevin Gimpel (2020). “Learning to Ignore: Long Document Coreference with Bounded Memory Neural Networks”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, pp. 8519–8526. DOI: 10.18653/v1/2020.emnlp-main.685.
- Ngoc Duyen Tanja Tu, Markus Krug, and Annelen Brunner (2019). “Automatic recognition of direct speech without quotation marks. A rule-based approach”. In: *DHd 2019 Digital Humanities: multimedial & multimodal. Konferenzabstracts. 6. Tagung des Verbands Digital Humanities im deutschsprachigen Raum e.V.* Ed. by Patrick Sahle. Frankfurt am Main, Germany, pp. 87–89. ISBN: 978-3-00-062166-6. DOI: 10.5281/zenodo.2600812.
- Don Tuggener and Manfred Klenner (2014). “A Hybrid Entity-Mention Pronoun Resolution Model for German Using Markov Logic Networks”. eng. In: *Proceedings of the 12th Edition of the Konvens Conference*. Hildesheim, Germany: Universitätsbibliothek Hildesheim, pp. 21–29. ISBN: 978-3-934105-46-1. DOI: 10.5167/uzh-99594.
- Asahi Ushio and Jose Camacho-Collados (2021). “T-NER: An All-Round Python Library for Transformer-based Named Entity Recognition”. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Ed. by Dimitra Gkatzia and Djamé Seddah. Online: Association for Computational Linguistics, pp. 53–62. DOI: 10.18653/v1/2021.eacl-demos.7.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 30. Long Beach, California, USA, pp. 5998–6008.
- Marc Vilain, John Burger, John Aberdeen, Dennis Connolly, and Lynette Hirschman (1995). “A model-theoretic coreference scoring scheme”. In: *Proceedings of the 6th Conference on Mes-*

- sage Understanding. MUC6 '95. Columbia, Maryland, USA: Association for Computational Linguistics, pp. 45–52. ISBN: 1558604022. DOI: 10.3115/1072399.1072405.
- Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma (2020). *Linformer: Self-Attention with Linear Complexity*. arXiv: 2006.04768 [cs.LG].
- Wei Wang, Bin Bi, Ming Yan, Chen Wu, Zuyi Bao, Jiangnan Xia, Liwei Peng, and Luo Si (2019). *StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding*. arXiv: 1908.04577 [cs.CL].
- Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli (2024). *Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference*. arXiv: 2412.13663 [cs.CL].
- Claire Warwick, Claire Warwick, Melissa Terras, and Julianne Nyhan (2012). *Digital Humanities in Practice*. Facet Publishing. ISBN: 9781856047661.
- Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. (2013). “OntoNotes Release 5.0 LDC2013T19”. In: *Philadelphia: Linguistic Data Consortium*.
- Max Wiechmann, Seid Muhie Yimam, and Chris Biemann (2021). “ActiveAnno: General-Purpose Document-Level Annotation Tool with Active Learning Integration”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations*. Ed. by Avi Sil and Xi Victoria Lin. Online: Association for Computational Linguistics, pp. 99–105. DOI: 10.18653/v1/2021.naacl-demos.12.
- Gregor Wiedemann, Steffen Remus, Avi Chawla, and Chris Biemann (2019). “Does BERT Make Any Sense? Interpretable Word Sense Disambiguation with Contextualized Embeddings”. In: *Proceedings of the 15th Conference on Natural Language Processing (KONVENS)*. Erlangen, Germany, pp. 161–170. URL: <https://arxiv.org/abs/1909.10430>.
- Gregor Wiedemann, Seid Muhie Yimam, and Chris Biemann (2018). “New/s/leak 2.0 – Multilingual Information Extraction and Visualization for Investigative Journalism”. In: *Social Informatics*. Ed. by Steffen Staab, Olessia Koltsova, and Dmitry I. Ignatov. St. Petersburg, Russia, pp. 313–322.
- Sam Wiseman, Alexander M. Rush, Stuart Shieber, and Jason Weston (2015). “Learning Anaphoricity and Antecedent Ranking Features for Coreference Resolution”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Chengqing Zong and Michael Strube. Beijing, China: Association for Computational Linguistics, pp. 1416–1426. DOI: 10.3115/v1/P15-1137.
- Sam Wiseman, Alexander M. Rush, and Stuart M. Shieber (2016). “Learning Global Features for Coreference Resolution”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kevin Knight, Ani Nenkova, and Owen Rambow. San Diego, California, USA: Association for Computational Linguistics, pp. 994–1004. DOI: 10.18653/v1/N16-1114.
- Wei Wu, Fei Wang, Arianna Yuan, Fei Wu, and Jiwei Li (2020a). “CorefQA: Coreference Resolution as Query-based Span Prediction”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky, Joyce Chai, Natalie Schluter,

- and Joel Tetreault. Online: Association for Computational Linguistics, pp. 6953–6963. DOI: 10.18653/v1/2020.acl-main.622.
- Xiaobao Wu, Thong Nguyen, and Anh Tuan Luu (2024). “A survey on neural topic models: methods, applications, and challenges”. In: *Artificial Intelligence Review* 57.2. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10661-7.
- Zhiyong Wu, Yun Chen, Ben Kao, and Qun Liu (2020b). “Perturbed Masking: Parameter-free Probing for Analyzing and Interpreting BERT”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online, pp. 4166–4176. DOI: 10.18653/v1/2020.acl-main.383.
- Patrick Xia, João Sedoc, and Benjamin Van Durme (2020). “Incremental Neural Coreference Resolution in Constant Memory”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, pp. 8617–8624. DOI: 10.18653/v1/2020.emnlp-main.695.
- Liyan Xu and Jinho D. Choi (2020). “Revealing the Myth of Higher-Order Inference in Coreference Resolution”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, pp. 8527–8533. DOI: 10.18653/v1/2020.emnlp-main.686.
- Zihang Xu, Ziqing Yang, Yiming Cui, and Zhigang Chen (2022). “HFL at SemEval-2022 Task 8: A Linguistics-inspired Regression Model with Data Augmentation for Multilingual News Similarity”. In: *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Ed. by Guy Emerson, Natalie Schluter, Gabriel Stanovsky, Ritesh Kumar, Alexis Palmer, Nathan Schneider, Siddharth Singh, and Shyam Ratan. Seattle, United States: Association for Computational Linguistics, pp. 1114–1120. DOI: 10.18653/v1/2022.semeval-1.157.
- Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel (2021). “mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou. Online: Association for Computational Linguistics, pp. 483–498. DOI: 10.18653/v1/2021.naacl-main.41.
- Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto (2020). “LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, pp. 6442–6454. DOI: 10.18653/v1/2020.emnlp-main.523.
- Seid Muhie Yimam, Chris Biemann, Richard Eckart de Castilho, and Iryna Gurevych (2014). “Automatic Annotation Suggestions and Custom Annotation Layers in WebAnno”. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Ed. by Kalina Bontcheva and Jingbo Zhu. Baltimore, Maryland, USA: Association for Computational Linguistics, pp. 91–96. DOI: 10.3115/v1/P14-5016.
- Seid Muhie Yimam, 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*. Ed. by Miriam Butt and Sarmad Hussain. Sofia, Bulgaria: Association for Computational Linguistics, pp. 1–6. URL: <https://aclanthology.org/P13-4001/>.
- Seid Muhie Yimam, Heiner Ulrich, Tatiana von Landesberger, Marcel Rosenbach, Michaela Regneri, Alexander Panchenko, Franziska Lehmann, Uli Fahrner, Chris Biemann, and Kathrin Ballweg (2016). “new/s/leak – Information Extraction and Visualization for Investigative Data Journalists”. In: *Proceedings of ACL-2016 System Demonstrations*. Ed. by Sameer Pradhan and Marianna Apidianaki. Berlin, Germany: Association for Computational Linguistics, pp. 163–168. DOI: 10.18653/v1/P16-4028.
- Wenzheng Zhang, Sam Wiseman, and Karl Stratos (2023). “Seq2seq is All You Need for Coreference Resolution”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, pp. 11493–11504. DOI: 10.18653/v1/2023.emnlp-main.704.
- Xiang Zhang, Junbo Zhao, and Yann LeCun (2015). “Character-level Convolutional Networks for Text Classification”. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Montréal, Canada, pp. 649–657.
- Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou (2025). *Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models*. arXiv: 2506.05176 [cs.CL].
- Yuanchi Zhang and Yang Liu (2022). “DirectQuote: A Dataset for Direct Quotation Extraction and Attribution in News Articles”. In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Ed. by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis. Marseille, France: European Language Resources Association, pp. 6959–6966. URL: <https://aclanthology.org/2022.lrec-1.752/>.
- Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du (2024). “Explainability for Large Language Models: A Survey”. In: *ACM Transactions on Intelligent Systems and Technology* 15.2. ISSN: 2157-6904. DOI: 10.1145/3639372.
- Shilin Zhou, Qingrong Xia, Zhenghua Li, Yu Zhang, Yu Hong, and Min Zhang (2022). “Fast and Accurate End-to-End Span-based Semantic Role Labeling as Word-based Graph Parsing”. In: *Proceedings of the 29th International Conference on Computational Linguistics*. Ed. by Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, pp. 4160–4171. URL: <https://aclanthology.org/2022.coling-1.365/>.