Bachelor Thesis

Efficient Wikipedia Entity Linking and Co-Reference Resolution in C++

Benjamin Dietrich

Examiner: Prof. Dr. Hannah Bast
Supervisor: Natalie Prange

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair of Algorithms and Data Structures

November 2^{nd}, 2022

Writing period

05. 08. 2022 – 05. 11. 2022

Examiner

Prof. Dr. Hannah Bast

Supervisor

Natalie Prange

DECLARATION

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I also hereby declare that my thesis has not been prepared for another examination or assignment, either in its entirety or excerpts thereof.

Place, Date	Signature

Abstract

A named entity is, for example, a person, product, or organization that its name can identify. Examples are 'Marie Curie', 'Snickers' or 'Microsoft'. Automatically recognizing and identifying named entities in texts is called entity linking. Automatically linking pronouns and entity types to the named entities they refer to in a text is called co-reference resolution. For example, the ship 'Titanic' has the entity type 'ocean liner' and can be referenced by the pronoun 'it'. The Python-based Wiki Entity Linker provides automatic entity linking and co-reference resolution on Wikipedia article texts. The system produces good results but has a long runtime. This thesis presents a redevelopment of the Wiki Entity Linker with a focus on efficiency. Two C++ versions of the Wiki Entity Linker were developed. Both systems have the same basic structure, but one uses an external Python-based natural language processing (NLP) library. The other system utilizes a complex set of rules. Both systems surpass the Wiki Entity Linker regarding runtime and result quality.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

There are many use cases for software-driven natural language processing, for example, automatic text summarization, chatbots, search engines, and many more. Often use cases contain models that can extract information from text. To perform their task, these models often need to be able to recognize named entities or capture the semantic content of a text. There are various definitions of what a named entity is in the literature. In this thesis, named entities are defined as proper nouns because the presented and evaluated systems in this thesis also consider only proper nouns. Proper nouns are names that designate, for example, persons, places, or objects. Examples of proper nouns are 'Marie Curie', 'Miami', or 'Taj Mahal'. On the other hand, nouns like 'scientist', 'city', or 'building' will not be considered named entities.

In order to train such information-extracting models, large data sets are needed. Text corpora with already annotated entity mentions are particularly well suited for such training. Wikipedia[1] is an excellent source for such a text corpus. The reason is that Wikipedia articles offer some advantages over normal texts due to their structure. In a Wikipedia article text, certain text passages are often already marked by the following tags:

- **Hyperlink Tag:** Links a named entity in the article text to its corresponding Wikipedia page. The hyperlink consists of two components, the target, and the text. The target is the link, which directly points to the Wikipedia page of

[1] `https://www.wikipedia.org/`

the marked named entity. The text represents the hyperlink in the article text. Example of a hyperlink: text

- **Bold Tag:** A bold tag marks the named entity the Wikipedia article is about. The bold tag only contains the text of the synonym. Example of a bold tag: text

- **Section Tag:** The section tag marks a new section in a Wikipedia article 'Section::::', followed by the title of the section. Example of a Section-Tag: Section::::Title

Wikipedia recommends that authors hyperlink the first mention of a named entity in the text[2]. If it is a particularly well-known named entity, so the hyperlink does not provide any added value for the reader, Wikipedia even recommends omitting the hyperlink. Therefore, named entities in Wikipedia articles are sometimes not marked by a hyperlink. In a high-quality text corpus, as many entity mentions as possible should be linked to the corresponding named entities. If an automated system performs the linking process, it is called entity linking [Section 3.1]. A knowledge base with additional information about the named entities is needed to annotate entity mentions that are not marked by a hyperlink. A suitable knowledge base for Wikipedia articles is Wikidata[3]. Wikidata is a free and open knowledge base that contains structured data about named entities. Both humans and machines can read Wikidata. Almost every Wikipedia page has an entry in Wikidata. Wikidata contains additional information about named entities. Many Wikipedia pages use Wikidata as a source of knowledge to populate the infoboxes[4] in the articles with data from Wikidata. Each entry in Wikidata has a unique identification number called 'QID'. A 'Q' as the prefix characterizes the id. A unique number follows the prefix.

[2] https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Linking
[3] https://www.wikidata.org
[4] https://en.wikipedia.org/wiki/Help:Infobox

For example, the famous ocean liner 'Titanic'[5] that sank in the Atlantic Ocean, has the QID 'Q25173'[6]. The QID can uniquely identify an entry in the knowledge base. Further mentions in the form of aliases, pronouns, or entity types, are annotated by utilizing the information contained in Wikidata. Entity types are type names that a named entity can have. For example, the Titanic is of type 'ocean liner'. The automated linking of pronouns and entity types in a text with the referenced named entities is called co-reference resolution [Section 3.2]. Annotating named entities, pronouns, and entity types in a text can often only be done satisfactorily by humans. Therefore, the task is mainly to work through the text word by word. In addition, each named entity mentioned in the text has to be annotated by a link or identification number. Since this is very time-consuming, there have been attempts to automate the whole process. However, high quality is essential to train models with the generated data. At the same time, an automated system should have an acceptable runtime to be usable in practice.

The entity linking and co-reference resolution process described in this introduction is formally defined in section 1.1. Section 1.2 explains the motivation for this thesis. Subsequently, the research question and the contributions this thesis offers are formulated in section 1.3. Finally, the main challenges encountered during development and the results archived are described in section 1.4.

1.1 Formal Problem Definition

Given a knowledge base KB, consisting of a set of unique named entities e.

$$KB = \{e_1, ..., e_n |\ \forall k, \forall l \in \mathbb{N}\ ((e_k = e_l) \Rightarrow (k = l)) \wedge n \in \mathbb{N}\}$$

[5] https://en.wikipedia.org/wiki/Titanic
[6] https://www.wikidata.org/wiki/Q25173

Each named entity contains an arbitrary set of attributes x and is identifiable by a unique identifier id.

$$e_{id} = \{x_1, ..., x_n \mid n \in \mathbb{N}\}$$

Let d be a document with an arbitrary set of individual texts t. Each text contains a set of entity mentions m and a set of other words w, which are not part of an entity mention. The entity mentions are not necessarily associated with a named entity from the knowledge base. The entity mentions also can be pronouns or entity types in the text.

$$t = \{\{m_1, ..., m_k\}, \{w_1, ..., w_l\} | k, l \in \mathbb{N}\}$$

$$d = \{t_1, ..., t_n | n \in \mathbb{N}\}$$

A system S, capable of performing entity linking and co-reference resolution in a text, now receives the document d as input and generates an output document d'. The output document d', now contains all texts from d. In addition to that, d' contains the start and end index of each entity mention in form of a tuple s, as well as the named entity $e_{id} \in KB$ mentioned at s in the text.

$$d' = \{(t_1, \{(s_{1_1}, e_a), ..., (s_{i_1}, e_b)\}), ..., (t_n, \{(s_{1_n}, e_c), ..., (s_{k_n}, e_d)\}) | a, b, c, d, i, k, n \in \mathbb{N}\}$$

The automated process of entity linking and co-reference resolution can be formulated as follows:

$$S(d) = d'$$

1.2 Motivation

In 2020, Natalie Prange and Matthias Hertel developed the Wiki Entity Linker[7] at the Albert-Ludwigs-University in Freiburg. The Wiki Entity Linker is based on the programming language Python and provides entity linking and co-reference resolution results with high quality. However, its runtime for one Wikipedia dump, with about 6.5 million Wikipedia pages, is roughly 30 days long. The Wiki Entity Linker could theoretically be considerably faster than 30 days since it is capable of using multiprocessing[8] for the processing of Wikipedia articles. However, a multithreading bug causes the RAM to fill with data unintentionally. If a complete Wikipedia dump is processed, the RAM will fill up before the Wiki Entity Linker can finish processing the dump. The bug does not occur when the Wiki Entity Linker runs on a single core. Therefore, the Wiki Entity Linker is currently only able to process a Wikipedia dump with one CPU core. The Wiki Entity Linker uses the natural language processing (NLP) library spaCy[9] [Section 3.3]. SpaCy makes it possible to identify entity mentions and pronouns. Although it is a comparatively fast library, the use of NLP libraries also extends the runtime of the Wiki Entity Linker. In addition, Python also has, on average, a 29.5 longer runtime than a comparable C++ program (Lion et al. [2022]). Therefore, developing a C++-based system with the same or better result quality and a shorter runtime is a rewarding research topic.

[7] `https://github.com/ad-freiburg/wiki_entity_linker`
[8] `https://docs.python.org/3/library/multiprocessing.html`
[9] `https://spacy.io/`

1.3 Contribution

This bachelor thesis is about the following research question:

How does the efficient use of the programming language C++ and the omission of the NLP library spaCy in the redevelopment of the Wiki Entity Linker, which was developed in 2020 by Natalie Prange and Matthias Hertel, affect the runtime and quality of the results of the system?

The following contribution can be derived:

- What runtime improvements can be achieved by using C++ compared to Python?
- How does multithreading additionally reduce the runtime?
- How does the omission of the natural language processing library spaCy, affect the runtime and quality of the results?
- How can the system be improved to further increase the quality of the results compared to the Wiki Entity Linker?

1.4 Approach

In order to achieve the goal of shortening the Wiki Entity Linker's runtime while maintaining the results' quality, several challenges had to be overcome. The Wiki Entity Linker uses spaCy, a library for natural language processing. The search for a suitable and similarly potent library for C++ remained unsuccessful. Therefore, spaCy was integrated into the C++ system in the form of the wrapper library spacy-cpp[10]. Since spaCy is Python-based, spacy-cpp is also Python-based at its core.

[10] `https://github.com/d99kris/spacy-cpp`

Due to the Global Interpreter Lock[11], which prevents the parallel use of Python programs on multiple threads, multithreading was not possible with spacy-cpp. In consequence, two versions of the C++-based Wikipedia Entity Linking and Co-Reference Resolution System (WELCORS$^{\text{C++}}$) were developed. One version uses spacy-cpp (WELCORS$^{\text{C++}}_{\text{spaCy}}$) and the other version does not use an NLP Library (WELCORS$^{\text{C++}}_{\text{non-NLP}}$). In order to be able to perform entity linking and co-reference resolution in a text without an NLP tool, a unique rule system for recognizing named entities and linking pronouns was developed. In addition, a data structure was developed that holds a complete Wikipedia article text and contains additional information about each word in the text. The data structure can be used to iterate word by word through the text. This data structure facilitates the rule-based search for named entities in the text.

The Wikipedia dump reading process of the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ was optimized for multithreading to reduce the runtime as much as possible. In addition, processing the Wikipedia articles on individual threads and outputting the results on individual output streams was also implemented. The WELCORS$^{\text{C++}}_{\text{spaCy}}$ has a runtime of almost seven days. Although this is significantly faster than the Wiki Entity Linker, the WELCORS$^{\text{C++}}_{\text{spaCy}}$ is still too slow. However, the WELCORS$^{\text{C++}}_{\text{non-NLP}}$, reaches an outstanding runtime of roughly seven hours on one CPU core and 53 minutes on 24 CPU cores for a full Wikipedia dump.

The quality of the results was measured using two benchmarks. Both WELCORS$^{\text{C++}}$ versions performed best in the benchmarks. In the more relevant Benchmark of the two, the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ achieved a F1-Score [Section 5.3.6] of 72.14% and the WELCORS$^{\text{C++}}_{\text{spaCy}}$ even reached 73.06%. The Wiki Entity Linker shows a F1-Score of 67.22%. The quality of the results was not only maintained but even exceeded. The improvements [Section 4.4] that were implemented specifically for the two WELCORS$^{\text{C++}}$ versions are the reason for the good results. Previous systems

[11] `https://wiki.python.org/moin/GlobalInterpreterLock`

have mostly focused on recognizing and annotating as many named entities in the text as possible. The improvements to the two WELCORS^{C++} versions additionally focus on avoiding as many incorrectly annotated named entities as possible.

The evaluation [Chapter 5] shows the superiority of a rule-based entity linking and co-reference resolution system over an NLP library-based variant. Although the WELCORS$^{C++}_{spaCy}$ has the best quality outcome, the difference to the WELCORS$^{C++}_{non\text{-}NLP}$ is small. In return, the WELCORS$^{C++}_{non\text{-}NLP}$ archives high-quality results in an unprecedented short runtime.

2 Related Work

2.1 Wiki Entity Linker

The starting point of this bachelor thesis is the Wiki Entity Linker[1]. No paper about the system has been published yet. So only the link to the GitHub repository can be attached in the footnotes. Since the basic structure of the Wiki Entity Linker has remained the same in the WELCORS^{C++}, more details are provided in chapter 4. Python is a dynamic and runtime-interpreted high-level programming language. Because of its clear syntax and the many available open-source libraries, Python is often used for natural language processing (NLP) tasks. The Wiki Entity Linker uses the NLP library spaCy [Section 3.3] for entity linking and co-reference resolution. The Wiki Entity Linker consists of the following three main components:

- **Entity Database (ED):** The entity database contains the information needed to identify named entities that appear in Wikipedia article texts. This information was obtained from Wikipedia and Wikidata. The information is saved in mapping files that are loaded at the system startup. For example, if a hyperlink recognizes a named entity, the QID is determined with the entity database via the hyperlink target. With the QID, further information such as entity types, aliases, or even gender can be determined from the database. This information can then be utilized to identify further mentions of the named entity in the text.

[1] `https://github.com/ad-freiburg/wiki_entity_linker`

- **Parser:** The Parser filters all bold and hyperlink tags from the Wikipedia article text. The bold tags are stored as synonyms for the named entity the Wikipedia article is about. The hyperlink targets are stored to retrieve a QID from the entity database and annotated later. All HTML tags and the Wikipedia article-specific section tag are removed and replaced with the tag texts.

- **Linking System:** The parsed text and all extracted information are then passed to the Linking System. The Linking System contains three linkers. Those three linkers are applied to the input text one after the other.

  Link Text Linker: Links other entity mentions based on the named entities already recognized by the hyperlinks, bold tags, and the title of the Wikipedia article. Other mentions are recognized in the text by the name of the named entities and the aliases of the respective named entities. The aliases are loaded from the entity database.

  Popular-Entities-Linker: The article text is processed again. Now more entities in the text are searched for which could not already be identified by the HTML tags. The Popular Entities Linker utilizes the NER [Section 3.3] tool of spaCy. The Popular Entities Linker recognizes potential named entities in the text and then tries to find a QID in the entity database for them.

  Co-Reference-Resolution-System: The last step of the linkage is the resolution of the co-references. The pronouns in the text are now linked with the entity mentions recognized so far. It also searches and links entity types in the text derived from the named entities recognized so far.

2.2 WEXEA: Wikipedia EXhaustive Entity Annotation

WEXEA promises exhaustive linking of the entities and their co-references in Wikipedia articles (Strobl et al. [2020]). To avoid unnecessary errors, WEXEA does not use an NLP library in its linking process. Instead, WEXEA uses a set of rules to detect named entities. WEXEA processes a Wikipedia article in four steps. First, mappings necessary to match named entities with a knowledge base are loaded. WEXEA loads necessary mapping files between each linking step. Then WEXEA uses the structure of Wikipedia articles, just like the Wiki Entity Linker, to detect and link named entities. To find more named entities in the Wikipedia article text, WEXEA uses its rule-based named entity recognition (NER). The rule-based NER of WEXEA recognizes named entities as follows. All capitalized words that are not stop words are recognized as named entities. Stop words often appear at the beginning of a sentence. They are not named entities but capitalized due to their position in the sentence.

If named entities are next to each other in the text, they are combined. In addition, lowercase words next to named entities are also checked to see if they form an alias from the mappings when combined with the named entities. Also, named entities are combined if they are connected by one of the words 'de', 'of', 'von', or 'van'. In the third step, the pronouns and entity types in the text are linked to the named entities annotated in the previous step. Finally, these co-references are linked to the named entities they reference. The last step in the process is the resolution of candidate conflicts. A conflict exists when several matching candidates are found in the text's mappings for a named entity. The conflict is resolved by selecting one of the candidates and annotating the named entity in the text. The quality of results and runtime of WEXEA is included in the evaluation.

2.3 Hedwig: A Named Entity Linker

Hedwig (Klang and Nugues [2020]) is a partially related system. Hedwig is also a system that performs entity linking on Wikipedia article texts. Like the other two systems presented above, Hedwig first extracts information from the hyperlinks and bold tags of the Wikipedia articles. This is done by converting the HTML version of the Wikipedia article into Docria layers(Klang and Nugues [2019]). Docria is a document model which analyzes a text in different layers. It splits the text into sentences and tokens. Docria also performs NER and Dependency Parsing on text. The usage of Docria in Hedwig is similar to the usage of spaCy in the Wiki Entity Linker. In the second step, the hyperlinks in the Wikipedia article are annotated with ids from the Wikidata knowledge base. In addition, further entity mentions are identified and linked in the text. However, Hedwig does not perform any co-reference resolution beyond that. It also links nouns in addition to the proper nouns defined as named entities. Furthermore, Hedwig does not publish its source code, which is why it could not be included in the evaluation of the systems.

3 Background

In this chapter, the background knowledge necessary for this thesis is discussed. The tasks of entity linking [Section 3.1] and co-reference resolution [Section 3.2] are explained in full detail. The functionality of the individual features of spaCy used in the WELCORS$^{\text{C++}}_{\text{spaCy}}$ is explained in section 3.3. The prefix search tree is presented in section 3.4.

3.1 Entity Linking

One of the tasks that all the systems presented in this thesis perform is entity linking. Entity linking is about recognizing as many named entities as possible in a text and annotating them with a unique identification number or link. The systems presented in this thesis do not annotate named entities in any text, specifically in Wikipedia article texts. A Wikipedia article often already provides some valuable information about the entities in the text that can be extracted and used. The system must recognize and annotate the rest of the named entities in the text. The following example from the Wikipedia article of Michelangelo[1] illustrates this:

***Michelangelo di Lodovico Buonarroti Simoni**, known simply as **Michelangelo**, was an Italian sculptor, painter, architect and poet of the High Renaissance. Born in the Republic of Florence, his work had a major influence on*

[1] `https://en.wikipedia.org/wiki/Michelangelo`

the development of Western art, particularly in relation to the Renaissance notions of humanism and naturalism.

In the above example, only the terms written in blue are already present in the text as hyperlinks. Clearly, they point to the Wikipedia page of the corresponding named entity. The hyperlinks can be utilized to identify the corresponding QID and annotate the named entity. The terms written in **bold** are within **bold tags** in the original Wikipedia article. Terms tagged in this way are considered synonyms for the named entity the Wikipedia article is about. In our example 'Michelangelo' is a synonym for 'Michelangelo di Lodovico Buonarroti Simoni'. The entity mentions in the form of tagged synonyms are annotated accordingly. The three named entities marked in red are not tagged in any way. Entity linking is also about recognizing these entities and annotating them accordingly.

3.2 Co-Reference Resolution

If two or more expressions refer to the same named entity, it is a co-reference. In natural language, co-references are quite common. For example, when pronouns like 'he', 'she' or 'it' are used to refer to a named entity. A co-reference can be replaced with the named entity it refers to without changing the sentence's meaning. Recognizing these pronouns and annotating them with the exact named entity they refer to is the second of the two objectives of the systems presented in this thesis. Also, entity types are annotated in addition to pronouns in the co-reference resolution. The following example is a passage from the Wikipedia article of the Hindenburg[2]:

As the airship passed over Munich on its second trial flight the next afternoon, the city's Lord Mayor, Karl Fiehler, asked Eckener by radio the LZ129's name, to which he replied "Hindenburg".

[2] `https://en.wikipedia.org/wiki/LZ_129_Hindenburg`

The named entities in the text are marked in blue were annotated by the entity linking. The pronouns and entity types in the text are marked in red. The entity types have to be recognized, and the pronouns have to be linked to the referenced named entities in the text. At the beginning of the sentence in the example, the word 'its' references the entity type 'the airship'. The Entity Type itself references the Hindenburg. The pronoun could also refer to the city of Munich. Correctly identifying the referred mention is the challenge of co-reference resolution.

3.3 spaCy

A powerful and widely used natural language processing library is spaCy[3]. It is based on the Python and CPython programming languages. There is no official version of spaCy for C++. This is why the wrapper library spacy-cpp[4] was used for the redevelopment. The wrapper spacy-cpp provides an interface for a real spaCy instance. The following tools from spaCy were utilized for both the Wiki Entity Linker and the WELCORS$^{\text{C++}}_{\text{spaCy}}$:

- **Named Entity Recognition (NER):** From a text, the NER extracts an iterable list of named entities. In addition, the named entities are, for example, classified as 'person', 'product', or 'work of art'. In total, spaCy's NER tool has 18 different classes. The Wiki Entity Linker and the WELCORS$^{\text{C++}}_{\text{spaCy}}$ ignore all text fragments of the classes 'cardinal', 'date', 'money', 'ordinal', 'percent' and 'quantity'.

- **Tokenizer:** Enables the segmentation of sentences into individual words and punctuations. The different segments are called tokens.

[3] `https://spacy.io/`
[4] `https://spacy.io/universe/project/spacy-cpp`

- **Part Of Speech (POS) Tagger:** SpaCy has different tags and labels like 'PROPN', 'DET', or 'NOUN' to classify the words in a sentence. The POS tagger, predicts the part of speech for each word in a text [Figure 1][5].

- **Dependency Parser:** Extracts the dependency parse of a sentence to represent its grammatical structure [Figure 1].

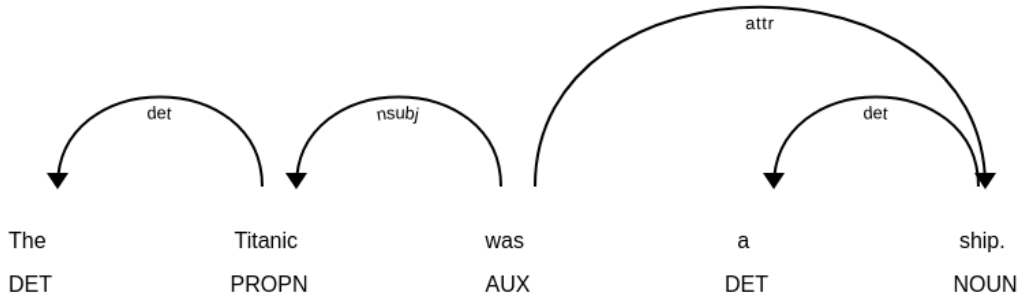

Figure 1: POS tagging and Dependency parse visualization of the sentence 'The Titanic was a ship.', created by the spaCy module displaCy. The Tags of the POS tagger are placed under each word of the sentence. The arrows visualize the dependency parse.

With those tools, it is possible to identify entity mentions in the article text that are not already marked as such with a hyperlink.

3.4 Prefix Search Tree

The prefix search tree (PST), also known as a trie, is a tree data structure that enables an efficient search in a large set of key strings in $O(n)$ (Laaksonen [2017]). The letters of the key strings form a chain of nodes. Each node contains one letter of the key string. Key strings with a common prefix share the same chain of nodes. The tree branches at the letter where two or more key strings differ. Figure 2 shows this behavior for the key strings 'Bag' and 'Bad' at the node with the letter 'a'. Key

[5] The spaCy module DisplaCy generated the visualization of figure 1. `https://spacy.io/universe/project/displacy-cpp`

strings do not have to consist of a single word. They can be composed of any number of letters and characters.

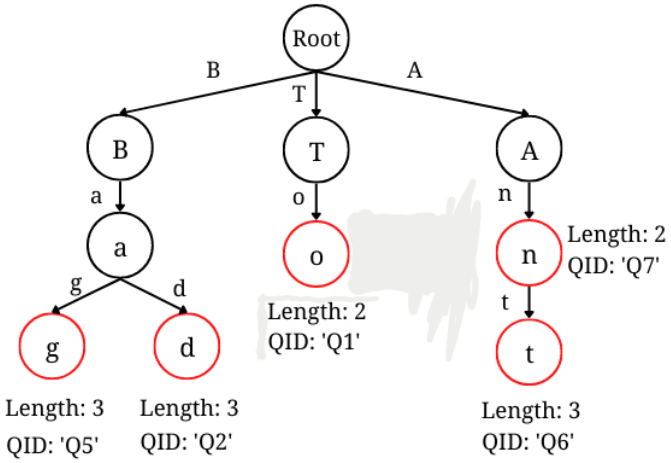

Figure 2: This prefix search tree contains the key strings 'An', 'Ant', 'Bad', 'Bag', and 'To'. Although these are not named entities according to the definition of this thesis, the key strings were chosen in order not to make the figure unnecessarily large. The red nodes represent an end node of a key string and contain the key string's length and the QID of the named entity referred to by the key string. The QIDs in this figure are fictitious.

The PST is utilized to detect key strings in the Wikipedia article text. For this purpose, the text is passed character by character into the PST filled with the searched key strings. The PST starts with a search run at the beginning of the text. Each time a space is passed to the PST, a new search run is started at the root node of the PST. The PST can perform several search runs at the same time. For each character that is entered into the search tree during the iteration through the text, all searches are continued by the one character entered. The search run is terminated if no child node with a matching character is found during a search run. If a search reaches an end node, the data in the end node is returned. An end node contains the length of the key string and the referenced QID. The length can be used to compute the start and end index of the named entity mentioned in the text. The QID will then be added to the entity mention before it is saved. A search continues even if an end node has been reached. Only when a leaf node has been reached or no suitable child

node is available the search run will be terminated. The PST always saves the first QID entered for a key string. The new QID is ignored if an attempt is made to save the exact key string with a different QID in the PST later. This rule is because the names of named entities that are persons are divided into first and last names. In order to correctly annotate the named entities if they are only named by their first or last name later in the text. An example is the following passage from the Wikipedia article by Tony Hawk[6]:

Tony Hawk was born on May 12, 1968, in San Diego, California, to Nancy (1924-2019) and Frank Peter Rupert Hawk, and was raised in San Diego. He has two older sisters, Pat and Lenore, and an older brother, Steve. As a child, Hawk was described as "hyperactive", and his mother stated that he was "so hard on himself and expected himself to do so many things.

The named entity a Wikipedia article is about is mentioned first in the text. For example, suppose Frank Peter Rupert Hawk has a Wikipedia page and an entry in Wikidata. Without the rule to ignore changes to already entered keys string, Tony Hawk's father Frank Peter Rupert Hawk would be linked every time 'Hawk' is mentioned in the text. The reason is that his father was mentioned after Tony Hawk in the text. Therefore the QID of the key string 'Hawk' would have been changed to the QID of his father.

[6] `https://en.wikipedia.org/wiki/Tony_Hawk`

4 Implementation

In this chapter, the implementation details of the redevelopment are presented. In section 4.1, each component of the WELCORS$^{\text{C++}}_{\text{spaCy}}$ and its workflow is described. In section 4.2, the reasons why a second version of the WELCORS$^{\text{C++}}$ without using an NLP library was developed and how this was possible is discussed. The section 4.3 is about how and where multithreading is implemented in the WELCORS$^{\text{C++}}_{\text{non-NLP}}$. Finally, the improvements to the quality of results implemented in both WELCORS$^{\text{C++}}$ versions are explained in more detail in section 4.4.

4.1 Components of the WELCORS$^{\text{C++}}_{\text{spaCy}}$

Since the WELCORS$^{\text{C++}}_{\text{spaCy}}$ is a redevelopment of the Wiki Entity Linker, it consists of the same main components. In order to efficiently detect entity mentions in the Wikipedia article text, a prefix search tree (PST) [Section 3.4] was implemented. The following subsections explain the functionality and interaction of the individual components in detail. Those components are the Linking System (LS) [Section 4.1.2], the Parser [Section 4.1.3], the Link Text Linker (LTL) [Section 4.1.4], the Popular Entities Linker (PEL) [4.1.5] and the Co-Reference Resolution System (CRS) [Section 4.1.6]. Besides differences in the efficiency, the component structure of the WELCORS$^{\text{C++}}_{\text{spaCy}}$ and the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ only differ a bit in the way the components interact. In the WELCORS$^{\text{C++}}_{\text{spaCy}}$ the Linking System also triggers the parsing process [Figure 3], whereas in the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ this happens before the Linking System is started

[Figure 4]. Besides the mentioned difference, everything explained in the following subsections also applies to the WELCORS$^{\text{C++}}_{\text{non-NLP}}$.

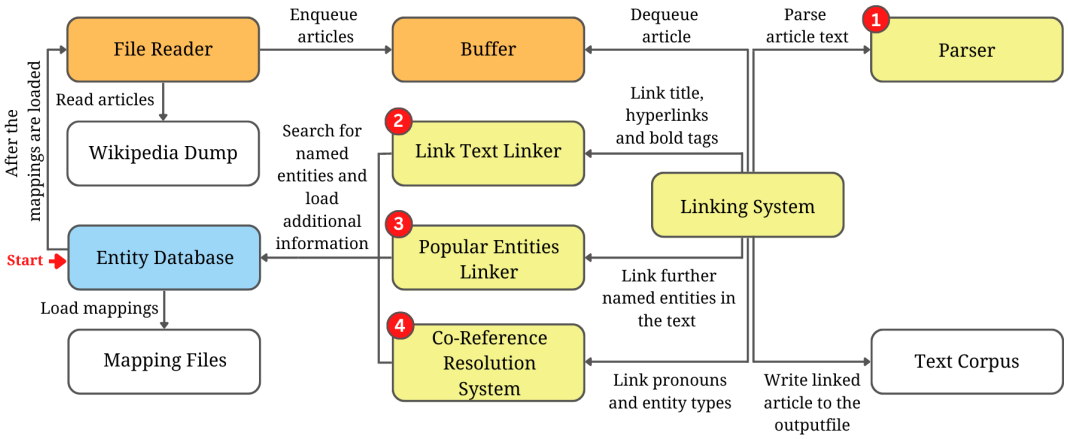

Figure 3: Shows the structure of the WELCORS$^{\text{C++}}_{\text{spaCy}}$. Files are white, the entity database is blue, system components responsible for reading and serving Wikipedia articles are orange, and the linking system is yellow.

4.1.1 Entity Database

To successfully link the entities of the Wikipedia article text, the same kind of entity database (ED) as in the Wiki Entity Linker is needed. The data used in the ED has been obtained from Wikidata and Wikipedia itself. The data was stored in mapping files loaded when the system started. The mapping files require about 9 GB of hard disk space. When the mappings are loaded, they require about 41 GB of RAM. The mappings are stored in unordered maps[1] and unordered sets[2], which are hash maps and hash sets. The hashing guarantees a constant-time complexity for search, insertion, and removal, which is crucial when dealing with multi-million different named entities and all kinds of additional information about each one of them. The following list is divided into the two origins of the mapping files.

[1] `https://en.cppreference.com/w/cpp/container/unordered_map`
[2] `https://en.cppreference.com/w/cpp/container/unordered_set`

Wikidata:

- *Aliases:* Alternative entity names used to find other entity mentions in the text.
- *Coreference Types:* Types of entities for which the Co-Reference Resolution System searches for in the text.
- *Demonyms:* Designations for the inhabitants or natives of a wide variety of regions. They are also recognized by the Popular Entities Linker.
- *Gender:* Gender of persons included in the knowledge base. It is used to determine, for example, which pronouns could refer to the respective entity.
- *Label:* The most common name for a named entity is known. This mapping is also utilized in the Popular Entities Linker.
- *Language:* Mapping of all languages. This is useful because languages are often not annotated by a hyperlink.
- *Site link:* Contains the site link count for each named entity. A site link count is a number that reflects the number of different languages into which the Wikipedia page of a named entity has been translated. The site link count determines how relevant a named entity is.
- *Wikipedia URL:* Maps the QID to the Wikipedia URL of the entities. This mapping is important for the Link Text Linker.

Wikipedia:

- *Acronyms:* Contains the acronyms of entities that are also searched for in the text.

- *Redirects:* Some Wikipedia pages have alternate links. For example, this can be the case if the name of a named entity or the original link of the Wikipedia page has changed. The redirect mapping helps identify the named entity even if the link is no longer correct.

4.1.2 Linking System

The Linking System (LS) controls the Parser [Section 4.1.3] and three linkers. Those linkers are the LTL [Section 4.1.4], the PEL[Section 4.1.5], and the CRS [Section 4.1.6]. First, the LS starts the Parser, which pre-processes the text. When the Parser is done, it stores the tag-free text and the information of the HTML tags in an article object. The article object also contains the title and the URL of the processed Wikipedia article. The article object will also contain all entity mentions when the linking process is done. The LS receives the article object and passes it to the LTL. After the LTL is done, the article object is passed to the PEL. Each time the LTL or the PEL detect an entity mention, they load the QID from the ED and pass it to the LS. The LS uses the ED to collect additional information about the named entity and stores all the names that can refer to the named entity in the text in a PST [Section 3.4]. Each time one of the two linkers is done, the LS uses the PST to search and link all entity mentions in the text.

The LS filters out span-wise overlapping entity mentions. Two entity mentions overlap if the span of one entity mention is within the span of the other. The shorter entity mention will be filtered out. At the end of the filtering process, the LS stores all remaining entity mentions in the article object. After the LTL and the PEL are

finished, the LS starts the CRS. The CRS now links pronouns in the article text to the entity mentions found by the LTL and the PEL. The CRS also links entity types in the text. After all three linkers are finished, the LS writes the article object to an output file in JSONL[3] format. The JSONL format contains a JSON[4] object in each line of the file. The output file of the Wiki Entity Linker contains an article object in JSON format in each line. The following four sections are about the Parser and the three linkers. There is a detailed explanation of how they process the Wikipedia article to detect entity mentions.

4.1.3 Parser

In order to detect the HTML tags in the article text, the Parser iterates through the entire Wikipedia article character by character. The Parser reads a new character every eight bits. Thereby the Parser increments a text index which is used to compute the start and end index of an entity mention in the text. In contrast to Python, the UTF-8 encoding is not built-in by default in C++. UTF-8 (Unicode Transformation Format in 8 bits) is a special character encoding of the Unicode Standard[5]. A block consisting of 8 bits is also called an octet in the technical literature. A UTF-8 encoded Unicode character consists of at least one octet and, at most, four octets. If the character consists of only one octet, the most significant bit is set to '0', and the remaining seven bits encode the character. If the character consists of $n > 1$ octets, the n most significant bits of the first octet are set to '1'. For all other $n - 1$ octets, the two most significant bits are set to '10' (Yergeau [2003]).

Since UTF-8 encoded characters can be up to 64 bits long, this leads to challenges in indexing each character correctly. The text is processed as a string and is iterated

[3] `https://jsonlines.org/ordered_set`
[4] `https://www.json.org/json-en.html`
[5] The Unicode Consortium, "The Unicode Standard – Version 4.0", defined by The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1), April 2003, `http://www.unicode.org/unicode/standard/versions/enumeratedversions.html#Unicode_4_0_0`.

UTF-8 Octett Sequence (Binary)
0xxxxxxx
110xxxxx 10xxxxxx
1110xxxx 10xxxxxx 10xxxxxx
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Table 1: Binary UTF-8 encoding. (Yergeau [2003])

character by character. When a UTF-8 encoded character is encountered in the text that consists of three octets, for example, the index is also incremented three times. The length calculation and indexing of strings, is made for ASCII encoding. In ASCII every character is encoded with 8 bits.

The UTF-8 encoding [Table 1] of each octet is utilized to still use strings to process the UTF-8 encoded text. First, the two most significant bits of each octet are checked. Then, the text index is incremented for each octet where the two most significant bits are not set to '10'. The two most significant bits are set to '10' when the octet is not the beginning of a character. This way, every character in the text is guaranteed to increment the text index exactly once. This method is utilized to compute the correct length of a string as well.

As already mentioned, text passages can be tagged in Wikipedia articles in three ways. The Parser reacts to these tags as follows when it detects them:

- **Hyperlink Tag**: Some symbols in links and, therefore in hyperlink targets are sometimes encoded in the so-called percent-encoding[6]. The encoding has a '%' as a prefix and continues with the hexadecimal representation of the ASCII symbol encoded in the URL. For example, '%2B' is the code for the '+'. In order to search for the hyperlink target in the database, the Parser first decodes all those symbols. Then the Parser checks if the hyperlink target differs from the hyperlink text. If this is the case, the differing hyperlink text is cached in

[6] `https://developer.mozilla.org/en-US/docs/Glossary/percent-encoding`

addition to the hyperlink target as an alias for the linked named entity. Finally, the hyperlink is replaced by the hyperlink text in the article text.

- **Bold Tag**: The bold tag text is stored as a synonym for the named entity the Wikipedia article is about, and the tag text also replaces the bold tag in the Wikipedia article text.

- **Section Tag**: The section tags are entirely removed from the Wikipedia article text, and only the section's title remains in the article text.

After the Wikipedia article text has been processed, the hyperlinks, synonyms, Wikipedia article title, Wikipedia article URL, and cleaned-up text are stored in an article object. Then, the article object is returned to the LS for further processing.

4.1.4 Link Text Linker

The LTL processes all named entities and synonyms extracted from the Wikipedia article's structure. The LTL checks whether a matching QID is available in the ED. Each time a QID could be determined, the LTL passes the named entity and its QID to the LS. The LS proceeds by filling the PST as already explained in section 4.1.2. In addition, each time a QID is found in the ED, the gender of the named entity is checked. If the named entity is a person, the name of the named entity is cached in a separate vector. After all named entities and synonyms have been processed, the vector with the names is iterated. The names are split into first and last names. The names are also individually inserted as aliases into the PST the LS uses to search the text for entity mentions. After the LTL is finished, the LS proceeds with searching the text for entity mentions based on the results in the PST the LTL produced.

4.1.5 Popular Entities Linker

The Popular Entities Linker (PEL) is responsible for recognizing all previously unrecognized entities in the article text. To recognize named entities, the PEL utilizes the NER tool of spaCy [Section 3.3]. With the NER tool, the PEL iterates over all named entities in the article text. As mentioned in section 3.3, some named entities are ignored based on the classification of the NER tool. When the PEL recognizes a named entity, it performs the same steps as the LTL [Section 4.1.4]. However, how and which entity mentions are recognized fundamentally differs from the LTL. The PEL checks for each entity mention, whether it is a language, a demonym, a label, or an alias. The PELstops as soon as one mapping contains the named entity. The entity mentioned is ignored if no QID is found in the ED. At the end of this process, the article text is parsed by the LS as previously described in section 4.1.2.

4.1.6 Co-Reference Resolution System

The Co-Reference Resolution System (CRS) is responsible for linking pronouns and entity types [Section 3.2]. For this purpose, the CRS uses spaCy's Tokenizer [Section 3.3] to iterate the Wikipedia article text one last time, word by word. Each word is checked to see if it is part of an entity mention that has already been linked. The CRS gets this information from a boolean map that is kept in the article object. The map is exactly as large as the text is long. Each character in the article text has its boolean value. The boolean map is updated every time an entity mention is added to the article object. In this case, the entire range of the entity mention will be set to true in the boolean map.

At any point during iteration, the CRS needs to know which entity mentions have been in the text. This information is needed to link the pronouns. In order to correctly match the gender-specific pronouns to the entity mentions in the text, the CRS needs to know the corresponding gender of each namend entity referred to by the entity

mentions. For this purpose, the CRS contains a vector of length five called entity cache (EC). The first three indexes cache the named entities recognized last in the text according to gender. At index zero, named entities of gender 'male' are cached. At index one, named entities of gender 'female' are cached. At index two, named entities with 'neutral' gender are cached. Index three caches the immediate last recognized named entity regardless of gender. Index 4 caches the last noticed named entity regardless of whether it was of gender 'male' or 'female'. Each time an entity mention is detected while iterating the text, the CRS uses the Dependency Parser [Section 3.3] of spaCy. The Dependency Parser is utilized to check whether the entity mentioned is the current sentence's subject. If this is the case, the CRS determines the gender of the named entity referred to by the entity mention. Subsequently, the named entity is cached at the corresponding index of the EC.

Every time a new section begins in the Wikipedia article text, the EC is reset. Two things are checked if the current word is not an already linked entity mention. First, whether the word is a pronoun. Second, whether it is the beginning of an entity type. If it is a pronoun, the gender of the pronoun is determined. Then the corresponding entity is obtained from the EC and linked to the pronoun. It will be tried to link a pronoun to the named entity the Wikipedia article is about if the entity cache at the index matching the gender is empty. However, the linking only is done if the gender of that named entity matches the pronoun. The beginning of an entity type is recognized by the presence of a 'The' or 'the' in the article text. If one of those two words is recognized in the Wikipedia article text, a prefix search tree (PST) [Section 3.4] prepared by the LS [Section 4.1.2] is utilized to check the following words. If an entity type is recognized, it is linked to the corresponding named entity. After the iteration of the text by the CRS is finished, the LS writes the results to the output file as described in section 4.1.2.

4.2 Replacement of the NLP library

The $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ is fully functional, but it still depends on the Python-based library spaCy. In the PEL, spaCy automatically recognizes entity mentions in the text. In the CRS, spaCy detects the subjects of each sentence. Only subjects are linked to the pronouns. To enable system-wide multithreading, Python had to be removed from the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$.

First, an attempt was made to find a C++-based NLP tool to replace spaCy. Unfortunately, there are hardly any tools in the C++ environment that are both up-to-date and have the same functionality as spaCy. The experiments with the C++-based NLP tool FreeLing[7] showed that the processing runtime of a Wikipedia article text by FreeLing is many times higher than that of spaCy. Thus, the use of FreeLing would not provide any added value.

The decision was made to replace spaCy in the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ by a complex rule system. In the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$, the second version of the redevelopment, exactly this was done. Subsequently, the system was equipped with worker threads. Each worker thread performs its own linking process independent of the other worker threads [Figure 4]. Each worker thread writes its results to a separate output file fragment via a separate output stream. At the end of the overall process, when all Wikipedia articles from the Wikipedia dump have been processed, the output file fragments are merged into one text corpus. Using different output streams impacts the overall runtime of the system when using more than four CPU cores. The worker threads do not have to wait their turn to write away the results, which makes the whole linking process faster.

How spaCy was replaced exactly is explained in the following subsections. First, section 4.2.1 describes how the Tokenizer [Section 3.3] was replaced by developing a data structure that contains the Wikipedia article text, including lots of information

[7] `https://nlp.lsi.upc.edu/freeling/node/30`

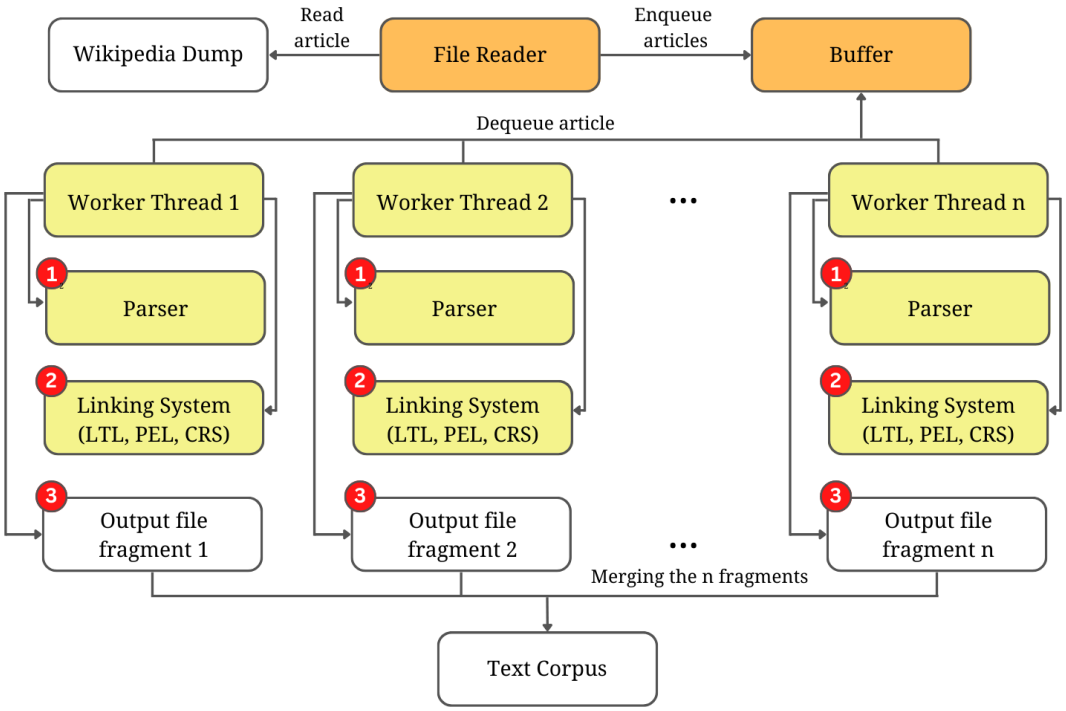

Figure 4: Shows the structure of the WELCORS$^{\text{C++}}_{\text{non-NLP}}$. Files are white, article serving components are orange, and article processing components are yellow. For clarity, the entity database (ED) has been omitted from the figure. The ED still is loaded at the start of the system and is accessed by the linkers as shown in figure 3.

about each word in the text. Subsequently, section 4.2.2 and section 4.2.3 are about the rule system that replaces spaCy in the PEL and the CRS.

4.2.1 Replacement of the Tokenizer

A text data structure that holds the text's information was implemented to replace the Tokenizer of spaCy. The text data structure itself contains sentence objects, which in turn contain word objects [Figure 5]. Each word object contains the following additional information about the word.

- The text of the word.

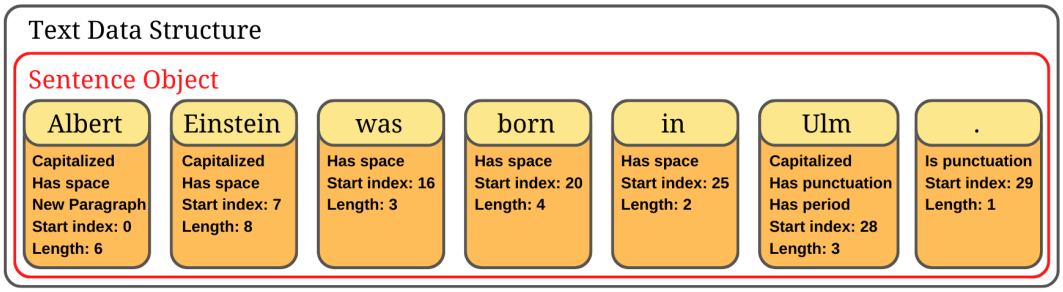

Figure 5: The text data structure can contain any number of sentence objects. The sentence objects can contain any number of word objects. Each word object contains additional information about the word in the sentence.

- (Boolean) Is punctuation.
- (Boolean) A space follows the word.
- (Boolean) A period follows the word.
- (Boolean) Any punctuation follows the word. (Example: '!', '?', ',').
- (Boolean) The word is the first word of a new paragraph.
- (Boolean) The word is an unlinkable entity (Set by the PEL).
- Start index of the word in the text.
- Length of the word

This information is used for entity recognition in the PEL and for the co-reference resolution in the CRS. The text data structure itself can output the text word by word. The Parser was adapted to fill the text data structure with individual words and the corresponding information. While it processes the text as before, it now also splits it word by word and passes each in the form of a word object to the text data structure. Each word object contains the additional information a word can have, as

mentioned above. The text data structure sorts the words into sentence objects and stores them. The text data structure can be used in the linking process to iterate word by word through the text. The additional information the text data structure provides is used to identify named entities in the PEL.

4.2.2 Rule Based Named Entity Recognition

A complex set of rules was implemented to replace the NER tool of spaCy. Since named entities are always capitalized, the words to be recognized must also be capitalized. Numbers are always considered to be capitalized. Furthermore, the words should be longer than one character, or if they have only one character, they must be followed by a period. A letter followed by a period is seen as the initial of a name. In addition, the word must not have been linked before by the LTL.

The PEL recognizes an entity mention in two phases. In phase 1, the first word of the entity mention is recognized, and in phase 2, all others. In phase 1, there are additional rules. The word must not be a pronoun or a stop word at the beginning of a sentence. Stop words are terms that frequently occur at the beginning of sentences. Usually, they are not part of an entity mention. If the stop words are at the beginning of a sentence, they are capitalized for grammatical reasons and considered part of an entity mention. Therefore, the entity database contains a list of stop words by which they are recognized. If a word meets all the rules, phase 2 begins.

First, it is checked whether the word recognized in phase 1 is a month, a number or disambiguation. If one of the above criteria applies, at least one other word must be added to the entity mention to be accepted. For example, 'January' will not be accepted, but 'January Jones'[8] is a legit named entity. Then, in phase two, each additional word that follows the first is checked and added to the entity mention if the following rules are met. In addition to the general rules, hyphens and the word 'of'

[8] `https://en.wikipedia.org/wiki/January_Jones`

are also allowed to connect capitalized words. If a word is recognized that no longer meets the rules, the end of the entity mention is reached. With the entity mention resulting from all these rules, the linking process is now performed as described in section 4.1.2.

4.2.3 Rule Based Co-Reference Resolution

The CRS of the Wiki Entity Linker and the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ uses spaCy's Dependency Parser to recognize the subject of a sentence and link pronouns accordingly. A rule was implemented that leads to good co-reference resolution results and replaces the Dependency Parser. [Section 5.3]. Intuitively, the rule states that an entity mention that is referenced by a pronoun is always mentioned first in the respective sentence.

In detail, it is implemented as follows. In the CRS, the text is iterated word by word one last time. Each time an entity mention is reached, its gender is checked. Then the entity cache is checked, whether an entity mention with the same gender was already present in the current sentence. If that is not the case, the just-checked entity mention is cached in the entity cache at the gender-corresponding index.

4.3 Multithreading

In order to make the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ even more efficient, multithreading was implemented. In section 4.3.1, more details about how multithreading reduces the loading time of the mappings are provided. The Pattern, implemented in the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ to process a single file on multiple threads, without the need to load the whole file at once completely into RAM, is explained in section 4.3.2. How the Wikipedia dump is processed on multiple threads, is illustrated in section 4.3.3.

4.3.1 Loading Mappings

Large mapping files loaded build the entity database. The total amount of data is about 9 GB. Transforming this amount of data into the entity database, which reaches a size of 41 GB, takes almost 13 Minutes [Table 2]. The loading of the mapping files has been distributed to multiple threads to reduce the runtime. Unfortunately, it is not possible to load all files in parallel. Some mapping files depend on other mapping files and can only be loaded when the mapping files they depend on are already loaded. Nevertheless, loading some of the files in parallel still provides a small runtime improvement of about 3 minutes [Table 2].

	Without Multithreading	With Multithreading
Runtime	12:46 min	09:36 min

Table 2: Loading runtime of the mapping files with and without using multithreading

4.3.2 Reading Input Data

A Wikipedia dump currently has a size of about 20 GB. The producer-consumer Pattern (Cooprider et al. [1974]) was implemented to not load the complete Wikipedia dump at once and still have enough Wikipedia articles in RAM, so the system runs smoothly. This Pattern contains a producer and an arbitrary amount of consumers. The producer fills a buffer with objects until it reaches a predefined maximum size. The producer thread is switched into an idle state if the maximum size has been reached. If a consumer takes an object from the buffer, it signals this to the producer thread. The producer is switched from the idle state back into an active state and checks whether the buffer size is below the maximum value. If the condition is met, the producer fills the buffer again. If not, the producer is switched back into an idle state until it is activated again by a consumer.

The consumers take objects from the buffer in parallel. In order to avoid a race condition, they are prevented from accessing the buffer simultaneously by mutual exclusion. If the buffer is empty and a consumer tries to take an object from the buffer, the consumer is switched into an idle state. The consumer is notified by the producer as soon as the producer has pushed an item into the buffer. The consumer becomes active and checks whether the buffer contains at least one object. The consumer takes the object from the buffer if this condition is met. Otherwise, it is switched back into an idle state.

Algorithm 1 Producer: Push Article to the Buffer Queue

```
while active do                                          ▷ Condition variable
    protect the queue from access by other threads;      ▷ Lock mutex
    if buffer queue size ≤ max size then
        break;
    end if
    release the protection of the queue;                 ▷ Release mutex
    switch into idle state;          ▷ Will be activated again by a consumer
end while          ▷ Producer is active and buffer queue size ≤ max size
push an article to buffer queue;
release the protection of the queue;                     ▷ Release mutex
notify a consumer;
```

Algorithm 2 Consumer: Pop Article of the Buffer Queue

```
while active do                                          ▷ Condition variable
    protect the queue from access by other threads;      ▷ Lock mutex
    if buffer queue size > 0 then
        break;
    end if
    release the protection of the queue;                 ▷ Release mutex
    switch into idle state;          ▷ Will be activated again by the producer
end while          ▷ Consumer is active and buffer queue size > 0
pops an article of the buffer queue;
release the protection of the queue;                     ▷ Release mutex
notifies the producer;
```

The file reader of the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ acts as the producer. It reads the articles one by one from the Wikipedia dump and uses a function [Algorithm 1] of the buffer to push the article into the queue of the buffer. The worker threads of the

$\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$, act as consumers. The worker threads parallel use a function [Algorithm 2] of the buffer to pop the front object of the queue.

The term 'busy waiting' describes a thread that repeatedly checks the same condition within a loop until it is fulfilled. This wastes system resources unnecessarily. Condition variables can be utilized to avoid busy waiting. Condition variables receive a mutex and a condition. The mutex is first locked, and then the condition is checked. If the condition is not met, the mutex is released, and the thread is switched into an idle state. After the thread is activated again by another thread, the mutex is locked again. Then, the condition is rechecked. This is repeated until the condition is fulfilled. In the pseudo code of this section, the condition variable is illustrated by the while loops in both algorithms.

4.3.3 Processing Wikipedia articles

In order to process the Wikipedia dump as efficiently as possible, the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ relies on multithreading for the entire processing. Each article is processed in a separate thread independent of the other threads. Two resources are accessed by all threads at the same time while processing the articles. One is the entity database (ED) [Section 4.1.1] and the other one is the output stream. Since nothing is written to the ED during the linking process, it is unnecessary to protect the maps and sets in the ED from concurrent accesses. The other resource accessed by all worker threads during the linking process is the output stream. Each worker thread has its own output stream and generates its own output file. The runtime of the system is reduced because the worker threads do not have to wait until it is their turn to write away their results. Attempts to use one output stream for all worker threads showed that the runtime could not be reduced further when more than four threads are used. The separate files of each worker thread are merged at the end of the linking process. Section 5.2 provides more details about the runtime.

4.4 Improvements

During the linking process in the PEL, entity mentions may be recognized but cannot be linked to a QID from the entity database. Usually, the reason is that the named entity the entity mention refers to does not yet have an entry in Wikidata. If it is a person, the person may be only named with the first or last name after the first mention in the text. It is frequently the case that the partial name is then also recognized as an entity mention. However, this partial name can then often be linked to a QID. Since the correct QID is not contained in the entity database, the QID that is now linked must be the wrong one. Therefore, 'unlinkable' entity mentions are marked in the text data structure as such, and their names are split. The parts of the names are ignored in the rest of the text to avoid the above-explained behavior. As an example, consider the following text passage from the Wikipedia article on the movie Closed Circuit[9]. The named entities 'Martin Rose' and 'Claudia Simmons-Howe' are both not available in the entity database:

When Erdogan's Defense Barrister commits suicide, Martin Rose is appointed by the Attorney General to replace him, joining Special Advocate Claudia Simmons-Howe. Due to national security concerns, Erdogan's case follows closed material procedures; Claudia represents Erdogan during closed sessions with a judge ruling what evidence is permissible for Martin to use in the open public trial.

Without caching the first and last names of the unlinkable named entities, the system would load incorrect QIDs for the partial names in our example. 'Claudia' would be linked to the first wife of the roman emperor Augustus[10] and 'Martin' to the city in northern Slovakia[11]. The unlinked entities are marked as such in the text data structure. The PEL does not link them, and the CRS later checks this in the text

[9] `https://en.wikipedia.org/wiki/Closed_Circuit_(2013_film)`
[10] `https://www.wikidata.org/wiki/Q237618`
[11] `https://www.wikidata.org/wiki/Q27001`

data structure while linking pronouns. While linking, the CRS checks the gender of the unrecognized entity mention based on the first name. If a pronoun refers to an unlinked entity mentioned in the text, the pronoun is not linked either. This improvement avoids many incorrect linked entity mentions and, therefor, improves the quality of the results.

5 Evaluation

This chapter deals with the evaluation of the memory requirements [Section 5.1], runtime [Section 5.2] and quality of the results [Section 5.3] of both WELCORS^{C++} versions. The runtime and quality of the results of WEXEA were also evaluated for comparison.

5.1 Memory requirements

The memory requirements were measured for a complete Wikipedia dump. This means that the memory requirement on the hard disk for the input and output file depends, of course, on the input file itself. For the mappings, the memory requirement always remains constant. The RAM requirements also remain constant regardless of the input file.

Memory	Mappings	Input File	Output File	Total
Hard Disk	9 GB	20 GB	52 GB	81 GB
RAM	41 GB	1.5 MB	500 KB	41 GB

Table 3: The memory requirements of of the Wiki Entity Linker and the two WELCORS^{C++} version. They all use the same mappings.

5.2 Runtime

In this section, the runtimes of WEXEA, the Wiki Entity Linker, the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ and the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ are evaluated. The tables list the constant runtime for loading the mappings separately from the actual processing time of the articles. Since WEXEA loads mappings several times between the processing steps, it does not output the times spent separately on them. These times have been measured manually. The measurement is, therefore, somewhat less accurate than it would have been if it had been measured automatically. Nevertheless, to differentiate between the runtime spent for loading mappings and the runtime of the linking process is important to compare the systems. Especially because the time spent loading mappings is constant and the time spent linking increases proportionally to the amount of processed Wikipedia articles. The tables are all sorted by the runtime of the linking process. However, each table also contains the total runtime. One thousand randomly selected articles from the entire Wikipedia dump were extracted and stored in a file to measure the runtime. For most measurements, unless otherwise specified, this file was used.

5.2.1 One Thousand Articles with One CPU Core

System	Mappings	Linking	Total
$\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$	00:09:44	**00:00:27**	00:14:15
WEXEA	00:04:12	**00:07:58**	00:12:10
$\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$	00:10:04	**00:13:12**	00:25:10
Wiki Entity Linker	00:14:13	**02:03:11**	02:17:24

Table 4: The runtime results of WEXEA, the Wiki Entity Linker and the two $\text{WELCORS}^{\text{C++}}$ versions. All systems have processed a file with one thousand articles. The times are formatted as follows: hh:mm:ss

In a direct comparison of the runtime for the linking process, the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ performs best with a runtime of 27 seconds [Table 4]. WEXEA has a linking process runtime of just under 8 minutes, which is surprisingly fast. WEXEA is just like the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ based on a set of rules instead of an NLP library. The two rule-based systems are ahead of the NLP library-based systems in terms of runtime. The $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ has a relatively long linking process runtime of barely 13 minutes, which in this case is solely due to the use of spaCy. In a direct comparison of the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ and the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$, the omission of the NLP library causes a linking process runtime reduction of 96,59%. The Wiki Entity Linker shows the highest runtime of all measuring systems, with a linking process runtime of about two hours. It was assumed that Python was the cause for the long runtime of the Wiki Entity Linker. This assumption is refuted by the short runtime of WEXEA, which is also Python-based. Another assumption was that the NLP library spaCy causes the long runtime. If that were the case, the spaCy-using $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ would have a similarly long runtime. The following calculation is intended to show that even the combination of Python and spaCy cannot be the cause of the long runtime. Lion et al. [2022] have shown that a Python program is, on average, 29.5 times slower than a comparable C++ program. The $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ has a runtime of 27 seconds on one thousand articles. Therefore, a comparable Python-based version of the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ would theoretically have roughly the following runtime:

$$27s \cdot 29.5 = 13{:}16 \text{ (mm:ss)}$$

From the measurements [Table 4], it is also known that spaCy causes a runtime difference of 12:45 (mm:ss) between the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ and the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$. Since both systems have the same code base, the 12:45 (mm:ss) can be considered the runtime of spaCy on one thousand articles. As spacy-cpp in the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ is also based on spaCy, the runtime difference can be directly added to the runtime of the theoretical Python version of the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$. This results in the

following calculation of the linking process runtime of a Python-based version of the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$, that is comparable in efficiency:

$$13{:}16 \text{ (mm:ss)} + 12{:}45 \text{ (mm:ss)} = 26{:}01 \text{ (mm:ss)}$$

There are inaccuracies in the calculation, and also, it is not said that a Python version would be precisely 29.5 times slower. The difference to the runtime of 2 hours of the Wiki Entity Linker still is high. Therefore, it is likely that the long runtime of the Wiki Entity Linker results from inefficient programming. The development of the Wiki Entity Linker was focused on the high quality of the results. Efficiency was neglected. Natalie Prange, who is one of the developers of the Wiki Entity Linker, confirmed this too.

5.2.2 One Thousand Articles with 24 CPU Cores

System	Mappings	Process	Reduction	Total
$\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$	00:09:58	**00:00:03**	88,89%	00:11:58
Wiki Entity Linker	00:14:11	**00:12:32**	89.84%	00:26:43

Table 5: The runtime results of Wiki Entity Linker and $\text{WELCORS}^{\text{C++}}$ using 24 CPU cores for multithreading. Both systems have processed a file with one thousand articles. The reduction indicates the percentage decrease of the runtime with 24 CPU cores, compared to the runtime previously measured in [Table 4] by using one CPU core. The times are formatted as follows: hh:mm:ss

Since the Wiki Entity Linker and the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ are capable of multithreading, the runtime for the processing of one thousand articles while using 24 CPU cores was measured [Table 5]. The amount of 24 CPU cores was chosen since this is the maximum amount available on the server used for the measurements.

When using multithreading, the Wiki Entity Linker performs better. As expected, the performance of the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ is nevertheless superior in comparison. Both systems reduce their processing time by almost the same percentage after increasing the number of CPU cores from one core to 24. During this measurement, it was observed how the Wiki Entity Linker gradually fills the RAM. Unfortunately, a measurement of the runtime of the Wiki Entity Linker with multithreading on a complete Wikipedia dump is not possible due to this bug.

5.2.3 Wikipedia Dump with Increasing Amounts of CPU Cores

Cores	Mappings	Linking	Reduction	Merging	Total
1	00:10:56	**06:18:03**	-	00:06:12	06:57:39
2	00:10:56	**03:17:00**	49.23 %	00:06:29	03:33:26
4	00:10:14	**01:32:39**	53.97 %	00:06:17	02:01:10
8	00:10:08	**00:47:45**	48.40 %	00:06:18	01:16:21
16	00:09:55	**00:28:51**	39.53 %	00:06:16	00:56:53
24	00:10:21	**00:25:01**	14.87 %	00:06:08	00:53:14

Table 6: The different runtimes of the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ using different amounts of CPU cores to process a Wikipedia dump with about 6.5 million Wikipedia articles. The times are formatted as follows: hh:mm:ss

To explore the potential of the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ the system was separately used to process an entire Wikipedia dump on different amounts of CPU cores [Table 6]. It was observed how the reduction in runtime decreased with the increase in the number of CPU cores. Between 1 and 8 cores, the runtime was always nearly halved while the number of cores was doubled. When doubling from 8 to 16 cores, a runtime reduction of only 39.53% was measured. Since a maximum of 24 cores can only be used simultaneously, no further doubling could be evaluated. An increase from 16 to 24 cores was expected to reduce the runtime by something between 20% to 25%.

However, the reduction was at 14.87%. The rate probably drops even further when more cores are used.

The decrease in runtime reduction with the increase of the used CPU cores cannot be explained at the moment. One assumption was that the hard disk could not quickly enough write away the amount of data generated by the CPU cores. However, it can be calculated that with a runtime of 25 minutes and the size of the output file of 52 GB, an average of $\frac{52000_{\text{MB}}}{(25 \cdot 60_s)} = 34.67_{\text{MB/s}}$ is processed by the hard disk. An average hard disk has a write speed of $80 - 160_{\text{MB/s}}$. Thus, the amount of data should not cause any runtime problems. The topic could be explored in more depth in a future work.

5.3 Quality

In this section, the benchmarks [Section 5.3.1] and the tool [Section 5.3.2] used to evaluate the quality of the results are presented. The metrics [Section 5.3.3] on which the evaluation is based and how they are calculated are explained. After that, the evaluation of the quality of the results of WEXEA, the Wiki Entity Linker and the two WELCORS$^{\text{C++}}$ versions follow [Section 5.3.7].

5.3.1 Benchmarks

In order to evaluate the quality of the results of the systems, two benchmarks were used. One is the Wiki-Ex benchmark by Natalie Prange and Matthias Hertel. In addition, a benchmark by Johanna Götz was used. Since the second benchmark has no official name, it is referred to as 'Four-Full-Articles' in the following. No paper has yet been published on either benchmark, so that no source can be provided at this point. However, both benchmarks were created at the Albert-Ludwigs-University of

Freiburg, and the Wiki-Ex benchmark can also be found in the GitHub repository[1] of the Wiki Entity Linker. Both benchmarks contain Wikipedia articles and additionally a ground truth for each Wikipedia article. The ground truth contains all entity mentions a system should predict if it is perfect. The entity mentions contain the named entity and the start and end index of the entity mention in the text. The start and end index is also called span. When the evaluated system has processed the benchmarks, and the linking results are available, the ground truth can be used in order to check the correctness of each linking result with the help of Elevant [Section 5.3.2].

Wiki-Ex contains 80 Wikipedia articles, from which certain text sections are benchmarked. The Wiki-Ex benchmark was explicitly developed for measuring the quality of the results of systems that perform entity linking and co-reference resolution on Wikipedia articles texts. The Wiki-Ex benchmark considers proper nouns, related pronouns, and entity types. In contrast, the Four-Full-Articles also includes nouns. Nouns are not supposed to be recognized by the evaluated systems of this thesis. Therefore, all systems perform worse in the Four-Full-Articles benchmark than in the Wiki-Ex benchmark. Nevertheless, the Four-Full-Articles benchmark was still considered to check if the ranking of the systems remains the same in both benchmarks.

5.3.2 Elevant

Elevant[2] allows a fine-grained evaluation of benchmark results of different linkers (Bast et al. [2022]). Elevant is used to evaluate and compare the quality of the results of WEXEA, the Wiki Entity Linker and the two WELCORS^{C++} versions [Section 5.3.7]. Elevant automatically calculates three different metrics to express the quality of the results of a system in percentage values. These three metrics are the Precision

[1] `https://github.com/ad-freiburg/wiki_entity_linker/blob/master/benchmarks/wiki-ex.benchmark.jsonl`
[2] `https://github.com/ad-freiburg/elevant`

[Section 5.3.4], the Recall [Section 5.3.5] and the F1-Score [Section 5.3.6]. In addition, Elevant offers an easy-to-use graphical interface with which all evaluated texts can be analyzed in detail.

5.3.3 Confusion Matrix

The Confusions Matrix has four fields [Figure 6] and is used to compare and classify predicted results with actual desired results. The Confusion Matrix can, therefore, also be used to compare the evaluated results with the ground truth of the benchmarks. If an entity mention of the predicted linking results is compared with an entity mention of the ground truth, they are considered to be matching if both contain the same span and named entity. If only one of the two criteria is met, they are not matching. The predicted entity mentions can be classified by three of the four fields of the confusion matrix:

- **True Positive (TP):** Both the span and named entity of the predicted entity mention match an entity mentions of the ground truth.

- **False Positive (FP):** No entity mention of the ground truth matches the predicted entity mention.

- **True Negative (TN):** The parts of the Wikipedia article text for which neither an entity mention was predicted, nor an entity mention is in the ground truth.

- **False Negative (FN):** A Entity mention of the ground truth, for which no matching entity mention was predicted.

This classification is used in the following three subsections to calculate the Precision [Section 5.3.4], the Recall [Section 5.3.5], and the F1-Score [Section 5.3.6].

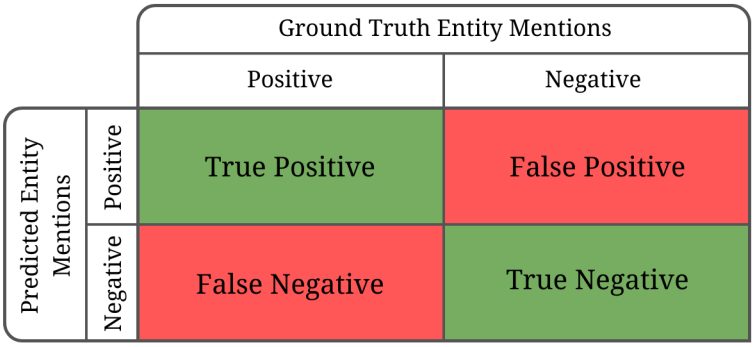

Figure 6: Shows a confusion matrix. The four Fields are the categories an entity mention can be classified by.

5.3.4 Precision

Precision is a metric that provides information about the percentage of correct predictions out of all predicted entity mentions. Two values are needed to calculate this value. On the one hand, the number of all correctly predicted entity mentions (TP). On the other hand, the number of all predicted entity mentions (TP + FP).

$$\text{Precision} = \frac{TP}{TP + FP} \cdot 100$$

A high Precision value can be deceptive because the Precision metric only includes the predicted entity mentions. Suppose only one entity mention out of 100 in the text was predicted. Further, the one predicted entity mention is a TP. The Precision would be at 100%, even though the system only predicted 1% of the entity mentions in the text.

5.3.5 Recall

The Recall metric provides information about how well the evaluated system correctly recognizes entity mentions. To calculate the Recall, the number of TP and the number of all entity mentions in the text (TP + FN) is needed.

$$\text{Recall} = \frac{TP}{TP + FN} \cdot 100$$

The Recall metrics can also be deceptive. Suppose a text consists of 200 words, and 100 words are entity mentions. Furthermore, suppose that for each of the 200 words, an entity mention was predicted by the system, and the 100 actual entity mentions in the text are TP. A Recall rate of 100% would be reached, even though only 50% of all predictions were correct.

5.3.6 F1-Score

The F1-Score is calculated as a third metric. The F1-Score combines the Precision and the Recall in one value. The F1-Score can be seen as a harmonic mean of Precision and Recall. The F1-Score is a mathematical value that cannot be described intuitively. For the calculation in percent, the TP, FP, and FN are used as follows.

$$\text{F1-Score} = \frac{TP}{TP + FP + FN} \cdot 100$$

The F1-Score has the greatest significance in the evaluation.

5.3.7 Results

Both benchmarks [Section 5.3.1] are included in each table, and the highest results in each column are highlighted. It should be noted that the Wiki-Ex benchmark is more relevant than the Four-Full-Articles benchmark. Therefore the systems are sorted according to their F1-Score in the Wiki-Ex benchmark.

	Wiki-Ex			Four-Full-Articles		
System	Precision	Recall	F1-Score	Precision	Recall	F1-Score
$\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$	**80.85**%	66.64%	**73.06**%	**52.35**%	**17.57**%	**26.31**%
$\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$	77.67%	**67.35**%	72.14%	47.38%	16.95%	24.97%
Wiki Entity Linker	72.48%	62.67%	67.22%	38.90%	11.63%	17.90%
WEXEA	76.12%	54.00%	63.18%	33.47%	6.05%	10.25%

Table 7: Results of the overall quality in the Wiki-Ex benchmark and the Four-Full-Articles benchmark. Elevant setting: Mention Types: All

In the overall evaluation of the two benchmarks [Table 7], both versions of the $\text{WELCORS}^{\text{C++}}$ perform best. In the Wiki-Ex benchmark, the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ benefits slightly from the higher precision of an NLP tool, which is expressed in a Precision score of 80.85%. However, the F1-Scores of the two $\text{WELCORS}^{\text{C++}}$ versions only differ by 0.92%. The lower precision of the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ is compensated by the higher Recall rate of 67.35%. The Wiki Entity Linker shows a about 5% lower F1-Score than the two $\text{WELCORS}^{\text{C++}}$ versions. The Wiki Entity Linker has, exactly like the other systems, the feature that splits names of named entities that are persons into first and last names to recognize them in the text. However, the feature works only partially in the current version. This leads to a lower Recall rate in the entity linking [Table 8] and, as a result, a lower Recall rate in the co-reference resolution [Table 9]. Furthermore, both $\text{WELCORS}^{\text{C++}}$ versions avoid some FP due to the improvements [Section 4.4]. WEXEA shows the lowest F1-Score of 63.18% among all tested systems.

The main reason for the poor overall results of WEXEA, is its performance in the co-reference resolution. WEXEA reaches only 31.83% [Table 9] in the F1-Score. WEXEA resolves co-references in a similar way to the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$.

	Wiki-Ex			Four-Full-Articles		
System	Precision	Recall	F1-Score	Precision	Recall	F1-Score
$\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$	**85.36**%	**81.36**%	**83.31**%	53.28%	**37.73**%	44.18%
$\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$	82.11%	**81.36**%	81.73%	**56.94**%	36.80%	**44.71**%
Wiki Entity Linker	78.80%	77.05%	77.92%	40.86%	26.49%	32.15%
WEXEA	81.08%	74.32%	77.55%	32.31%	12.99%	18.53%

Table 8: Results of the entity linking quality in the Wiki-Ex benchmark and the Four-Full-Articles benchmark. Elevant setting: Mention Types: Entity: Named

	Wiki-Ex			Four-Full-Articles		
System	Precision	Recall	F1-Score	Precision	Recall	F1-Score
$\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$	69.44%	**59.29**%	**63.97**%	31.07%	36.71%	33.65%
$\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$	**72.08** %	56.13%	63.11%	**51.85**%	**39.16**%	**44.62**%
Wiki Entity Linker	64.50%	50.99%	56.95%	34.09%	20.98%	25.97%
WEXEA	66.25%	20.95%	31.83%	50.00%	13.64%	21.43%

Table 9: Results of the co-reference resolution quality in the Wiki-Ex benchmark and the Four-Full-Articles benchmark. Elevant setting: Mention Types: Coref: All

WEXEA processes the whole article and caches the most recent entities. For every entity that WEXEA encounters, it loads the gender from the knowledge base Yago3 (Mahdisoltani et al. [2014]). WEXEA always links the last found entities of the respective genders to the pronouns that appear in the text up to the next entity. This distinguishes the CRS of WEXEA from the CRS of the $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$. $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ always links the first appearing entity mention of a gender in a sentence to the corresponding pronouns [Section 4.2.3]. This difference significantly impacts the resulting quality [Table 9].

The Four-Full-Articles benchmark shows the same ranking as the Wiki-Ex benchmark in the overall evaluation. In summary, the results of the Four-Full-Article benchmark confirm the results of the Wiki-Ex benchmark. The good performance

of the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ in entity linking is explained by the fact that it links any capitalized entity mention in the text based on the rules explained in section 4.2.2. Thus, it generally links more named entities in the text than the two systems that use spaCy for named entity recognition. However, the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ loses precision as a result.

6 Conclusion

In this thesis, two C++-based versions of a redevelopment of the have been presented. Both versions basically meet the intended goal of reducing the runtime of the Wiki Entity Linker and at least maintaining the quality of the results. In fact, both WELCORS$^{\text{C++}}$ versions exceeded the resulting quality of the Wiki Entity Linker. The WELCORS$^{\text{C++}}_{\text{spaCy}}$ shows a slightly better result quality compared to the WELCORS$^{\text{C++}}_{\text{non-NLP}}$, because it utilizes the NLP library spaCy. However, this comes at the cost of a significantly longer runtime compared to the WELCORS$^{\text{C++}}_{\text{non-NLP}}$. In comparison to the WELCORS$^{\text{C++}}_{\text{spaCy}}$, the Wiki Entity Linker and WEXEA, the runtime results of the WELCORS$^{\text{C++}}_{\text{non-NLP}}$ were remarkable. The WELCORS$^{\text{C++}}_{\text{non-NLP}}$ allows to process of a complete Wikipedia dump into a high-quality text corpus within an excellent runtime. The redevelopment fulfills the expectations of the project and transforms the well-designed but slow Wiki Entity Linker, into a practical, usable Wikipedia entity linking and co-reference resolution system, on the basis of C++.

6.1 Future Work

In a future work, more time could be invested in the analysis of language. For example, better rules for recognizing the subject of a sentence could be researched, or the recognition of entities could be refined. These rules could then be implemented and contribute to the quality of the results of theWELCORS$^{\text{C++}}_{\text{non-NLP}}$. Possibly the

resulting quality of the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$ could be reached or even exceeded since the two systems already differ by only 0.92% [Table 7]. Estimated time: 8-10 weeks

If, in the future, an NLP library based on C++ with similar runtime and quality as spaCy should become available, spaCy could be replaced by the new library in the $\text{WELCORS}^{\text{C++}}_{\text{spaCy}}$. The replacement would allow multithreading in the $\text{WELCORS}^{\text{C++}}_{\text{NEW-NLP}}$. As a result, the runtime of the $\text{WELCORS}^{\text{C++}}_{\text{NEW-NLP}}$ could be reduced. However, this is not possible today since there is no comparable C++-based NLP Library available. Estimated time: 1-2 weeks

Another approach to reduce the runtime of the $\text{WELCORS}^{\text{C++}}$ versions could focus on the constant times of each run. The construction and destruction of the entity database account for 22 minutes of each program run, regardless of the size of the input file. On the one hand, the mapping files could be adapted in such a way that any calculations performed during the construction of the entity database are already included in the mappings. For example, decoding all Wikipedia URLs. Also, some mapping files depend on other mapping files. This means that they can only be loaded when the other mapping file is already contained in the entity database. An example of a dependency would be the label mapping. The fact that labels can refer to several named entities. In order to be able to uniquely assign a named entity later in the text, the entry in the entity database that has the highest site link count [Section 4.1.1] is always selected. So there are entries in the label mapping which are never used for linking. The mapping files could be cleaned from such unused entities. This would reduce the file size and would make it possible to load more files in parallel since the dependency is also removed by the adjustment. With adjusted mapping files, the system loses the possibility to update the files directly by downloading the current state of Wikipedia and Wikidata. It would be conceivable to write a system that automatically downloads the data and generates already adapted mapping files. On the other hand, an attempt could be made to accelerate the destruction of the entity database by directly destruct maps that are no longer needed after a certain

point in the program. Estimated time: 6-8 weeks

The $\text{WELCORS}^{\text{C++}}_{\text{non-NLP}}$ shows a steadily decreasing runtime reduction with increasing CPU core quantity. Investigating this and possibly developing a solution strategy could improve the runtime of the system when using more than eight CPU cores. It would first be necessary to check which cause is responsible for this since this is unknown at the moment. Estimated time: 6-8 weeks

7 Acknowledgments

First of all, I would like to thank Natalie Prange, who provided me with helpful advice and support during the writing of the bachelor thesis.

I would also like to thank Prof. Dr. Bast for giving me the opportunity to write this bachelor thesis at her chair and for examining it.

My special thanks go to my sister-in-law Carolin Ank, who proofread my thesis with so much dedication.

I would also like to thank my family, who supported me where they could during my studies and especially in the intense last phase of this bachelor thesis.

Bibliography

Hannah Bast, Matthias Hertel, and Natalie Prange. Elevant: A fully automatic fine-grained entity linking evaluation and analysis tool. *arXiv preprint arXiv:2208.07193*, 2022.

Lee W. Cooprider, F Heymans, Pierre-Jacques Courtois, and David Lorge Parnas. Information streams sharing a finite buffer: other solutions. *Inf. Process. Lett.*, 3 (1):16–21, 1974.

Marcus Klang and Pierre Nugues. Docria: Processing and storing linguistic data with Wikipedia. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 400–405, Turku, Finland, September–October 2019. Linköping University Electronic Press. URL `https://aclanthology.org/W19-6148`.

Marcus Klang and Pierre Nugues. Hedwig: A named entity linker. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4501–4508, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL `https://aclanthology.org/2020.lrec-1.554`.

Antti Laaksonen. Competitive programmer's handbook. *Preprint*, 5:244–245, 2017.

David Lion, Adrian Chiu, Michael Stumm, and Ding Yuan. Investigating managed language runtime performance: Why JavaScript and python are 8x and 29x slower than c++, yet java and go can be faster? In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 835–852, Carlsbad, CA, July 2022.

USENIX Association. ISBN 978-1-939133-29-40. URL https://www.usenix.org/conference/atc22/presentation/lion.

Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *7th biennial conference on innovative data systems research*. CIDR Conference, 2014.

Michael Strobl, Amine Trabelsi, and Osmar Zaiane. WEXEA: Wikipedia EXhaustive entity annotation. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1951–1958, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.240.

François Yergeau. Utf-8, a transformation format of iso 10646. Technical report, 2003.