10.01.2025 - Bachelor Thesis - Department of Computer Science

# Citation Recognition in Legal Documents

## Named Entity Recognition with Transformer-Based Deep Learning Methods


| | |
|---|---|
| Authors: | Mirio Eggmann<br>Jasmin Fitz<br>Dario Glasl |
| Advisor: | Prof. Dr. Mitra Purandare |
| Co-Advisor: | Benjamin Plattner |
| Practice Partner: | Swisslex - Schweizerische Juristische Datenbank AG |
| Co-Examiner: | Marcel Amsler (Sanitas Management AG) |
| Proofreader: | Urs Baumann |

# Abstract

This bachelor thesis develops a transformer-based system for citation recognition in legal documents. It is intended to replace an existing solution based on regular expression matching, which is hard to maintain and does not generalize to new citation formats.

The task is divided into two steps: One model classifies citations in legal texts into CASELAW, LAW and LITERATURE. A second model identifies the parts of these citations, such as COURT, DATE, ARTICLE, etc.

The existing regex-based solution serves as data source and benchmark for the first step of recognizing citations. 10 million text chunks containing legal citations are used to fine-tune a *Google BERT base multilingual (uncased)* model. It achieves a test recall of 96.9%. On a manually annotated dataset, it scores a recall of 74.33%, while the regex-based solution achieves 72.15%.

Sparse training data poses a challenge for recognizing parts of a citation. Few-shot prompting with an LLM yields good results, but experiments show that it is prohibitively slow in practice. Therefore, knowledge distillation is used to fine-tune a *DistilBERT base (uncased)* model via the supervision of Llama 3.3 70B. The 1′044 times smaller DistilBERT achieves a similar performance to Llama. The model scores a test recall of 99.11%. On a manually annotated dataset, it achieves a recall of 93.37%.

The final .NET-based solution allows bulk processing of documents and provides a web interface for user interaction. FastAPI is used to serve the fine-tuned models. The results are stored in an MS SQL database.

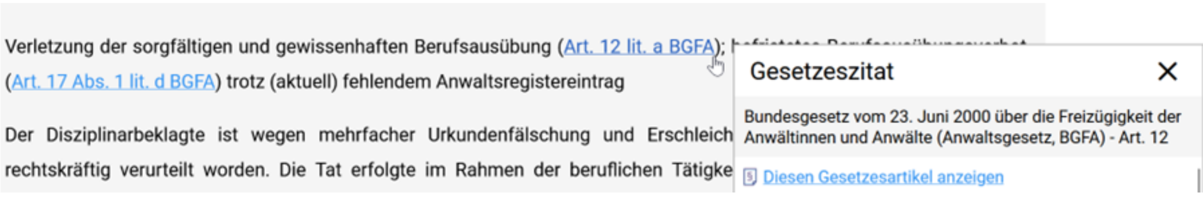
The proposed solution offers many advantages besides the at par performance with the regex-based system. The solution is more robust and can handle deviations such as typing errors and new citation formats more easily. Moreover, the cherry on top is the improved performance in recognizing parts of a citation with more granular labels.

# Management Summary

## Initial Situation

This bachelor thesis develops a system for recognizing citations in legal documents.

It is carried out in collaboration with Swisslex. They provide an online research platform for legal documents. Most of these legal documents reference legal literature or judgments. To facilitate research, these referenced documents are linked and can be accessed with one click. An example of such a reference can be seen in the following figure. In order to link documents, the citations and their specific parts need to be identified in the text.




Example of a linked citation in a legal document

## Current Solution

The current citation recognition in Swisslex is automated using regular expression (regex) matching. This is a method for finding patterns in text through predefined rules. This approach has its limitations. Since it uses strict patterns, minor deviations such as typographical errors or new citation formats are not recognized. The system is time-consuming and error-prone to maintain, with over 300 regexes stored in the database and dynamically generated in code. To make matters worse, the original developers of this system are no longer available and no documentation exists.

## Goal

The goal of this project is to develop a new AI-based system for citation recognition. The system should be able to recognize citations in legal documents as precisely as possible. This includes the identification of their sub-parts, such as court, year, page number, etc. Documents in the languages German, French, English and Italian should be supported. By using an AI model, the system should be more flexible and easier to maintain than the current regex-based solution.

## Procedure

First, the available data from the Swisslex database is analyzed and a literature review on related work is conducted. The data is then prepared as training and test data for the AI models. Different models are trained, evaluated and compared with the current regex-based solution. The best performing models are chosen and integrated into the new citation recognition system.

## AI Models

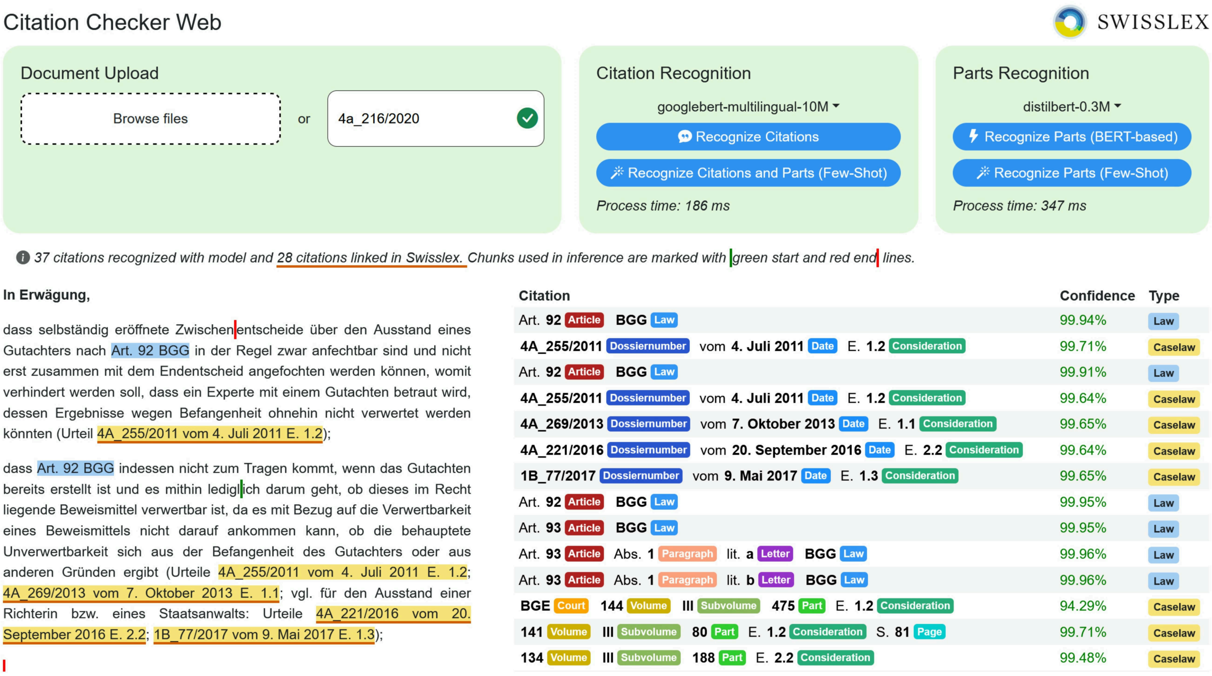
The new citation recognition system consists of two AI models: one trained on legal documents to identify citations, and another one trained on citations to recognize their parts. Together, they promote precise and fast recognition of citations within legal documents.

They achieve both very good results in the evaluation. The model for recognizing citations matches the performance of the regex-based solution. The model that recognizes the specific parts of a citation cannot be directly compared to the regex-based solution. A new set of fine-grained part types is introduced, since the types that the existing solution uses are too vague. This makes a direct comparison impossible. However, the AI model correctly recognizes the new part types of a citation 92% of the time.

When compared against the same model that OpenAI's ChatGPT is based on (GPT-4o), the new citation recognition system performs better. It recognizes citations in a text 15% more accurately.

## Testing via Web Application

To test the citation recognition system, a web application called *Citation Checker Web* is developed. The web application allows the user to upload a document or to search for a document available in the Swisslex database. The recognized citations and their parts are displayed in the user interface. In the next figure, a screenshot of the web application is shown.




Screenshot of the web application *Citation Checker Web*

## Bulk Processing of Documents

Swisslex processes around 3000 documents each night. To support this, a console application called *Citation Processor* is developed. It allows to recognize citations and their parts in a large number of documents automatically. The results are stored in a database. The new system is able to process those 3000 documents in approximately 30 minutes.

## Improvements over the Current System

The new AI-based system significantly improves the process of citation recognition. Within the document, it recognizes the citations equally well as the current regex-based solution. Additionally, it is more flexible and can handle typographical errors or changes in citation formats. Since the rules are not manually defined, it does not require manual maintenance. Also, more fine-grained parts are recognized. A web application facilitates evaluating the system continuously. The new system is built so that it can be easily extended and maintained in the future. When, for example, a better AI model gets available, it can be re-trained on the existing data. This takes only a few hours.

# Table of Contents

# 1 Introduction

The bachelor thesis develops a system that can recognize citations of legal documents, such as laws, judgments, or legal literature. It is carried out in collaboration with Swisslex as practice partner. This introduction gives an overview of the project and its goals.

## 1.1 Practice Partner

Swisslex is one of the most comprehensive legal research platforms in Switzerland. Every day, over 1500 legal professionals use the search and analysis tools to find the information they need for their work. The platform offers a wide range of legal information, including case law, legislation and legal literature. There are nearly one million interlinked documents available. Customers of Swisslex are mainly law firms, courts, authorities, and companies. The platform is used for legal research, drafting legal opinions, and preparing court cases. [1]

## 1.2 Initial Situation

The content available on Swisslex is provided by different publishers in the form of documents. A document may be a judgment, a book, a journal, or another legal medium. In these texts, there are often references to other documents, for example when citing a law or judgment. For these references, Swisslex directly links the cited documents. This allows users to navigate to the referenced document with one click. An example of such a reference can be seen in Figure 1.

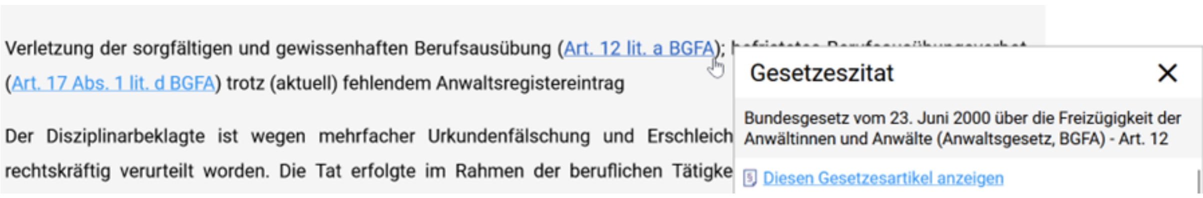



Figure 1: Screenshot of a citation reference

Currently, the citations in the documents are recognized manually as well as using regular expression matching. The process is explained in detail in chapter 3.1.4.2. This method has been used to identify about 60 million references in the entire Swisslex database. However, the process is very difficult to maintain. New or alternative citation formats have to be added manually. This is a time-consuming process and can lead to errors. Furthermore, the current solution is not able to handle complex citations or deviations from the standard citation format.

A previous attempt in Swisslex used the Flair framework [2] to recognize citations. A Swisslex employee worked on the prototype between November 2021 and April 2022. Different experiments were conducted, including the `T5Model`, `SequenceTagger`, `TaskClassifier` and `TextClassifier` models from Flair. The models were trained on a dataset based on the existing known citations.

The prototype was able to recognize citations in legal documents with an F1 score of 91%. Despite the good results, the prototype was not integrated into the production workflow. It could recognize simple citations which occur often, but had problems with complex citations and edge cases.

## 1.3 Task Description

The goal of this bachelor thesis is to recognize citations and their parts (work, page, paragraph, etc.) in legal documents using a modern transformer style NLP approach. The identified citations and parts should be stored in the database.

### 1.3.1 Research

The first step is to survey the state of the art in the field of NLP and to identify suitable approaches for the task. Different approaches should be evaluated and compared.

### 1.3.2 Software

The most promising approach should be implemented. The result should be a software which:

- **recognizes references** in a German, French, English or Italian text with the help of a specially trained or fine-tuned model
- recognizes references of laws, judgments and journals (this work does not focus on books and other legal media, but they may be considered when evaluating the model)
- **identifies its parts** (e.g. court, page, consideration, etc.) (at least if the regex that has been used to recognize the corresponding citation is available, as it is the case for judgments and journals)
- **saves these** in a suitable form together with their sources (e.g. database, Elastic Index, json, …). This should allow to later link the original document and the cited document, if both are available in the Swisslex database. The mapping to the referenced document is, despite of the original task description in appendix section F.7, not part of the bachelor thesis. This can be seen in the meeting minutes from the 07. and 10. October 2024, in appendix section F.6.2.

### 1.3.3 Evaluation

The resulting solution is to be evaluated on a test set of documents from the Swisslex database. The evaluation should also include a comparison with the existing solution. A successful model is similar or better in performance to the existing solution, but more robust against slight deviations, new patterns, typographical errors or context-based nuances.

The following points should be considered:

- **Accuracy**: The solution should be able to recognize references and identify its parts correctly.
- **Robustness**: The solution should be able to handle typographical errors, deviations from the standard format, and context-based nuances.
- **Throughput**: The solution should be able to process a large number of documents in a short time.

The main focus of the system is accuracy and robustness. Although real-time processing is not required, the system should still be able to handle about 3000 documents overnight. It should have a high recall. False positives are less of a concern, as they will be filtered out in a later step. However, that is outside the scope of this thesis.

### 1.3.4 Deployment

The software should be deployable and easy to integrate into the existing Swisslex infrastructure. Since the current *Production Tool* will be replaced in the near future, it is not necessary to integrate the solution into this tool. Instead, an alternative way to start the process should be provided, e.g. by a recurring job. This should process a list of documents and store the results.

Additionally, a simple user interface should be provided that allows the user to upload a document, start the processing and view the output. The UI should be accessible via a standard web browser.

## 1.4 General Conditions

This bachelor thesis is written at the OST - Eastern Switzerland University of Applied Sciences in Rapperswil as part of the bachelor program in computer science. The thesis is advised by Prof. Dr. Mitra Purandare and co-advised by Benjamin Plattner.

According to the module description [3], the thesis should demonstrate the ability to solve problems using engineering methods and has a conceptual, theoretical and practical component. In addition, there is an oral examination of approximately 30 minutes after submission of the thesis. The successful completion of the thesis results in 12 ECTS points, which corresponds to a workload of approximately 360 hours per person.

The start date is 16 September 2024 and the submission deadline is 10 January 2024 at 17:00. The exact module description can be found in the appendix section F.8.

# 2 Requirements

This chapter describes the requirements of the system for citation recognition. They are divided into functional and non-functional requirements.

## 2.1 Functional Requirements

The functional requirements are presented through use cases, as depicted in Figure 2.

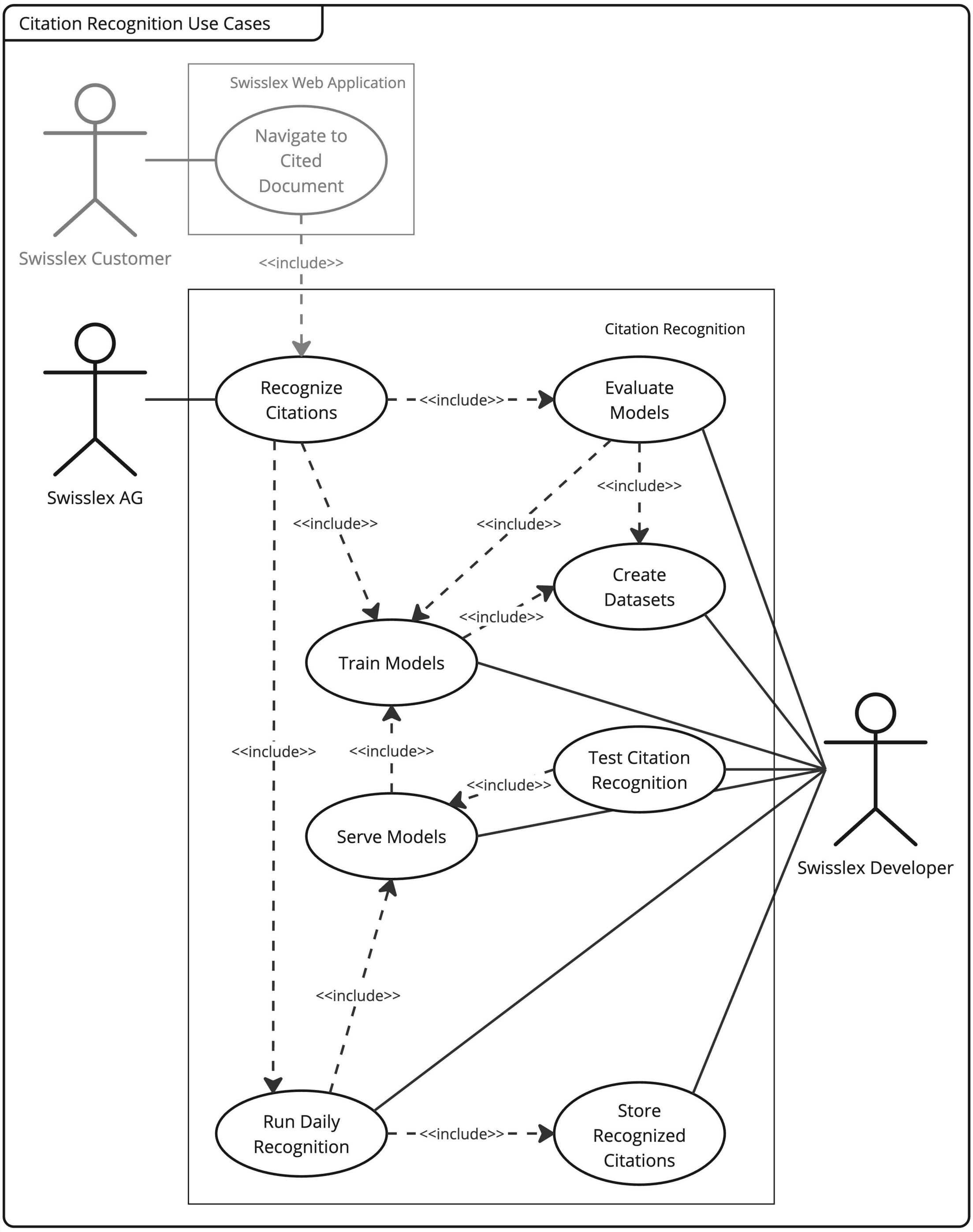



Figure 2: Use case diagram

### 2.1.1 Actors

Three main actors are involved:

**Swisslex customer** Users of the Swisslex platform conducting legal research. Their activities include searching for and reading legal documents.

**Swisslex AG** The organization responsible for providing the Swisslex platform.

**Swisslex developer** Employees at Swisslex responsible for the platform's development, including the citation recognition system.

### 2.1.2 Use Cases

This section describes the use cases relevant to the citation recognition system.

**Navigate to Cited Document** A Swisslex customer needs to easily navigate to a cited document, for example, by clicking on a citation. This feature enhances user experience and simplifies accessing the original sources of information. **Note:** This use case is an existing feature of the Swisslex platform and is only listed here for completeness. It is not part of this thesis.

**Recognize Citations** Swisslex AG seeks an automated solution to recognize citations in legal documents. This reduces the need to manually enter citations into the database. The recognized citations form the basis for navigating to the referenced documents via citation links.

**Train Model** A Swisslex developer must be able to train models to recognize citations in legal documents. Training models with modern machine learning techniques aligns with the industry's shift towards automated Named Entity Recognition (NER) tasks.

**Evaluate Model** A Swisslex developer needs to evaluate the performance of different models. Evaluation ensures the selection of the most effective model for production.

**Create Datasets** To train and evaluate models, a Swisslex developer needs datasets with texts based on legal documents and annotated citations.

**Test Citation Recognition** A Swisslex developer requires a visual interface to manually test the citation recognition system. This feature is critical for verifying that the system works as intended.

**Serve Models** A Swisslex developer wants to deploy trained models to a server for inference. Serving models enables the citation recognition system to process legal documents.

**Store Recognized Citations** The citation recognition system must store identified citations in a database. This allows linking documents to their cited sources and supports features like navigation.

**Run Daily Recognition** A Swisslex developer must schedule the citation recognition system to run daily. Regular processing ensures that the database remains updated as new documents are added.

## 2.2 Non Functional Requirements (NFR)

In this section, the non-functional requirements (NFR) are described.

| **ID** | **NFR-01** Performance |
|---|---|
| **Requirement** | A single legal document (e.g. judgment) should be processed within a reasonable time frame. |
| **Scenario** | The system processes a single legal document (e.g. judgment) and successfully identifies citations with its parts (`COURT`, `DATE`. `CONSIDERATION`, etc.). The user expects the document to be processed within 7 seconds. |
| **Stimulus** | A new document is sent to the inference endpoint for citation recognition. |
| **Reaction** | The software accurately identifies the citations and processes the document within the given time frame (7 seconds). |
| **Measures** | • Choose a scalable and efficient NLP model that can process documents quickly<br>• Optimize the processing pipeline for speed |
| **Level** | Must have |

Table 1: NFR-01 Performance

| **ID** | **NFR-02** Scalability |
|---|---|
| **Requirement** | The performance of citation recognition should be scalable and still accurate with increasing document throughput. |
| **Scenario** | The system processes a batch of legal documents (e.g. judgments or journals) and successfully identifies citations with its parts (`COURT`, `DATE`. `CONSIDERATION`, etc.). The user expects that 3,000 documents are processed within 6 hours. |
| **Stimulus** | A new batch of documents is uploaded, and the citation recognition process begins. |
| **Reaction** | The software accurately identifies the citations and processes the batch within the given time frame (6 hours). |
| **Measures** | • Use a fine-tuned or specially trained NLP model for multilingual citation recognition<br>• Optimize processing with parallelization and scalable infrastructure |
| **Level** | Must have |

Table 2: NFR-02 Scalability

| **ID** | **NFR-03** Compatibility |
|---|---|
| **Requirement** | The solution must be deployable within the Swisslex infrastructure. |
| **Scenario** | The system is deployable and integrates with the existing infrastructure, with an alternative process to start batch document processing. |
| **Stimulus** | The system is deployable and starts processing new documents. |
| **Reaction** | The citation recognition job can be started which processes the documents and stores the results in a compatible format. |
| **Measures** | • Design for integration into Swisslex infrastructure<br>• Offer alternative deployment methods, e.g. recurring jobs or batch processing<br>• The output format should be compatible with the existing solution |
| **Level** | Must have |

Table 3: NFR-03 Compatibility

| | |
|---|---|
| **ID** | **NFR-04** Reliability |
| **Requirement** | The solution should be robust against variations, new patterns, and typos. |
| **Scenario** | The system is evaluated against a test set of documents, and its robustness is compared to the existing regex-based solution. |
| **Stimulus** | Citations in German, French, English, and Italian texts include variations, typos, and new patterns. |
| **Reaction** | The software identifies citations with similar or better performance compared to the regex-based solution, showing resilience against deviations. |
| **Measures** | • Fine-tune the model with diverse datasets<br>• Continuously improve recognition accuracy through iteration |
| **Level** | Must have |

Table 4: NFR-04 Reliability

| | |
|---|---|
| **ID** | **NFR-05** Security |
| **Requirement** | The solution should ensure data privacy and security. |
| **Scenario** | The system processes proprietary legal documents and must ensure data privacy and security. |
| **Stimulus** | Legal documents are uploaded to the system for citation recognition. |
| **Reaction** | The software processes the documents and stores the results securely, ensuring data privacy and confidentiality. |
| **Measures** | • Comply with data protection regulations<br>• Implement a solution that runs on premise or in a secure cloud environment |
| **Level** | Must have |

Table 5: NFR-05 Security

| | |
|---|---|
| **ID** | **NFR-06** Usability |
| **Requirement** | Easy to use testing software for citation recognition. |
| **Scenario** | The user runs citation recognition on a document. |
| **Stimulus** | The user interacts with the software by providing a document and viewing the results. |
| **Reaction** | The software processes the document and displays the identified citations and its parts in a user-friendly format. |
| **Measures** | • Design a user-friendly interface for document upload and result display<br>• Provide feedback on the processing status and results |
| **Level** | Should have |

Table 6: NFR-06 Usability

# 3 Domain Knowledge

This chapter explores the environment in which the project is conducted. This includes the Swisslex system with its current system for citation recognition as well as the general domain of legal citations.

## 3.1 Swisslex Environment

The following sections describe the Swisslex environment. This includes an explanation of the documents and their processing workflow. The current system for citation recognition and its challenges are discussed.

### 3.1.1 Swisslex Web Platform

The Swisslex web platform https://www.swisslex.ch is a legal information platform that provides access to a wide range of legal documents and is the main product of the Swisslex company. The website offers different features, the most important of them being the search function and the display of legal documents. The platform is used by legal professionals to find and analyze legal documents.

Figure 3 shows an example of the usage of the Swisslex platform. The user searches for the phrase *"missbräuchliche Kündigung"* in combination with *"Ferien"* and *"Krank"*. As result, the platform displays a list of documents containing the search terms.

In this example, the search results in 286 hits. If necessary, the user can now filter the results according to various criteria, such as document type, publication date or publication. A number of tools make it possible to further analyze the documents found, for example by viewing a citation network, translating the document or highlighting search terms.

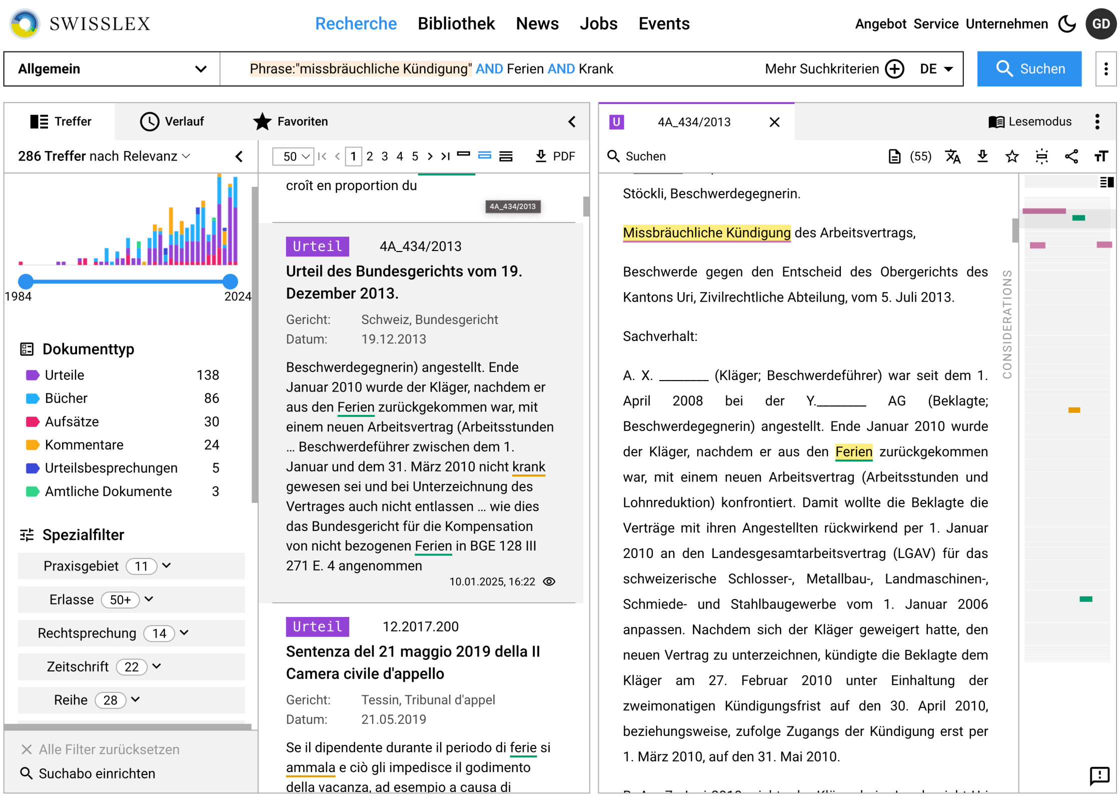



Figure 3: Screenshot of the Swisslex platform [4]

### 3.1.2 Document Processing

As mentioned in chapter 1.2, several publishers provide their content to Swisslex in the form of documents. There are different ways in which these documents arrive at Swisslex: some are uploaded by the publishers themselves, others are crawled from the Internet. An example of a Swisslex document can be seen in appendix section F.2.1. The following types of documents exist:

- Case law / Judgment
- Case law Review
- Journals
- Commentary
- Essay
- Book

Once they arrive at Swisslex, they are converted into a uniform XML format by an internal service. Afterwards, they are enriched with additional metadata using the internal *Production Tool*. During this process, citation references are recognized and marked up. In the end, the processed documents are stored in a Microsoft SQL Server Database, as well as in an Elastic database. The process is visualized in Figure 4.

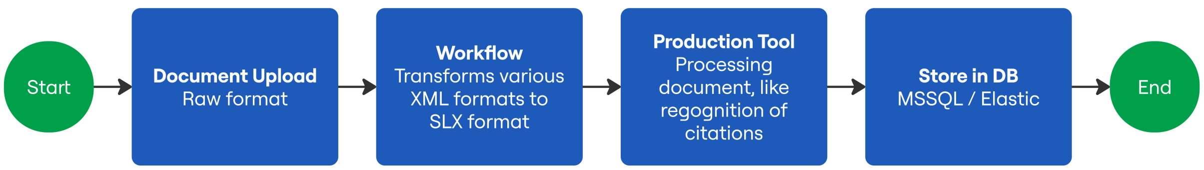



Figure 4: Flowchart of the production workflow

Every night, about 3′000 documents are (re-)indexed in approximately 6 hours. This updates the search index with the latest changes in the documents. During this process, the citation references are also re-scanned by the regex solution and stored in the database.

### 3.1.3 Domain Model

The Swisslex domain relevant to this bachelor thesis includes various types of legal documents, such as laws, judgments, and other forms of legal literature. These documents are accessible on the Swisslex platform and may include citations referencing other documents also available within Swisslex. The entities and their relationships are illustrated in the domain model shown in Figure 5.

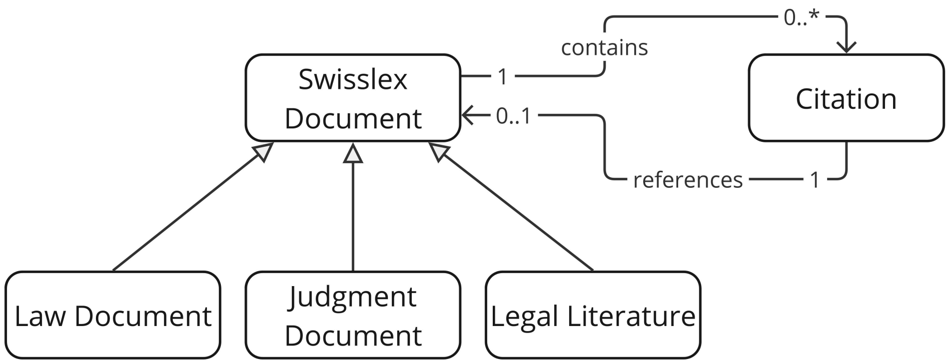



Figure 5: Domain model of the Swisslex environment

### 3.1.4 Production Tool

The *Production Tool* is an internal software application in Swisslex. It is used to process the documents that are uploaded to Swisslex. It is a Windows Forms app created in 2011 with C# and still runs today with the .NET Framework version 3.5. Swisslex employees operate the *Production Tool* and use it to enrich documents with additional metadata and save them to the database. The current implementation of the citation recognition (regex-solution) is part of the *Production Tool*. Swisslex intends to replace the *Production Tool* in the near future, because it is outdated and hard to maintain. This bachelor thesis is a first step into this direction.

#### 3.1.4.1 Citation Linking

To enable documents to be linked, the citations in the texts are detected and mapped to the corresponding documents in the Swisslex database. To mark the citations in the documents, they are enclosed in `<citation>` tags. Each tag contains the type of the citation and the citation ID. This is a unique identifier that links to the database record of the citation.

In Code-Snippet 1 there is a shortened section of a document before processing. In Code-Snippet 2, the same section is shown after processing. The citations are now enclosed in `<citation>` tags with the corresponding citation ID.

```
<text>
    <p align="justify">la Cour de justice du canton de Genève (C/3981/2022,
ACJC/389/2024).</p>
</text>
```

Code-Snippet 1: Text excerpt before processing - judgment 4A_240/2024

```
<text>
  <p align="justify">la Cour de justice du canton de Genève (<citation type="asset"
citationid="b9a1bb9e-2790-4c4e-a994-8e5756f12d90">C/3981/2022</citation>, <citation
type="asset"      citationid="73e3eacb-c99c-4028-a4bc-b68257d2c3e4">ACJC/389/2024</
citation>).</p>
</text>
```

Code-Snippet 2: Text excerpt after processing - judgment 4A_240/2024

#### 3.1.4.2 Current Solution for Citation Recognition

The current solution for citation recognition is based on regular expressions (regex). When processing a document, the *Production Tool* uses these regexes to recognize the citations in the text. Most of the regexes are stored in the database. However, some are also dynamically generated in the code of the *Production Tool*. This combination makes it impossible to use the existing regexes for citation recognition without using the *Production Tool*.

An example of the stored regexes in the database can be seen in Figure 6. A complete example of a regex is shown in chapter 3.3.3.

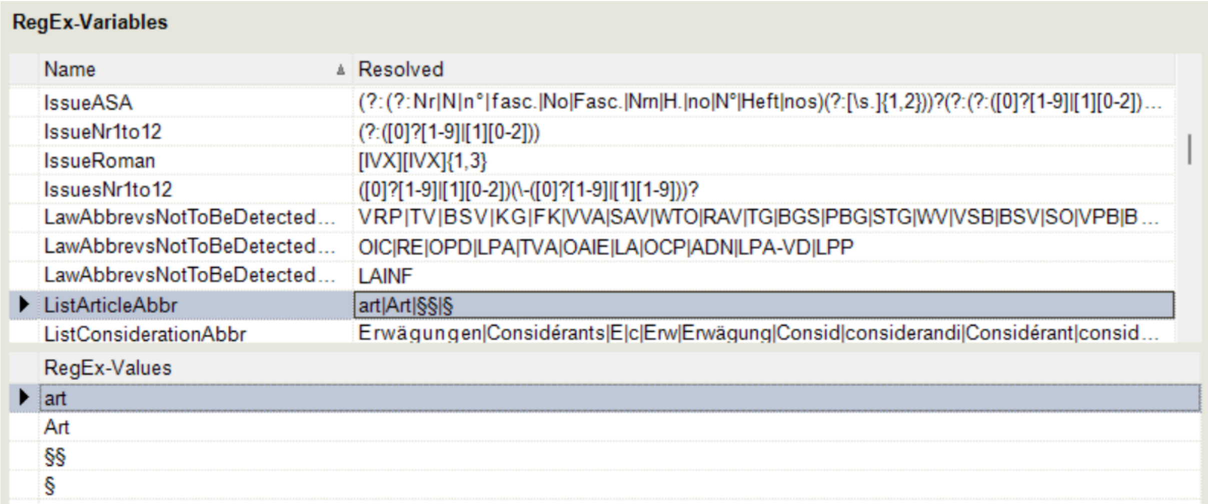



Figure 6: Regex stored in the database

#### 3.1.4.3 Database Schema

An excerpt of the database schema relevant for the citation recognition process is shown in Figure 7. All documents have an entry in the `AssetBase` table. All citations are stored in the `CitationBase` table. The recognized citation is stored in a field called `CitationPassage` in the `CitationBase` table.

Since there are different ways to cite the same document, the `JurReference` table is used to normalize the citations. If a normalized form of a citation can be generated, the corresponding `CitationBase` entry is linked to the `JurReference` entry. If the citation links to a document available in Swisslex, the `JurReference` links back to the `AssetBase` entry of the document.

When the citation was recognized by the regex solution, the `JurReference` links additionally to the `JurReferenceRegEx` table. This table contains the regex that was used to recognize the citation.

These two tables `AssetBase` and `CitationBase` are linked through multiple other tables. They are omitted in this figure for simplicity.

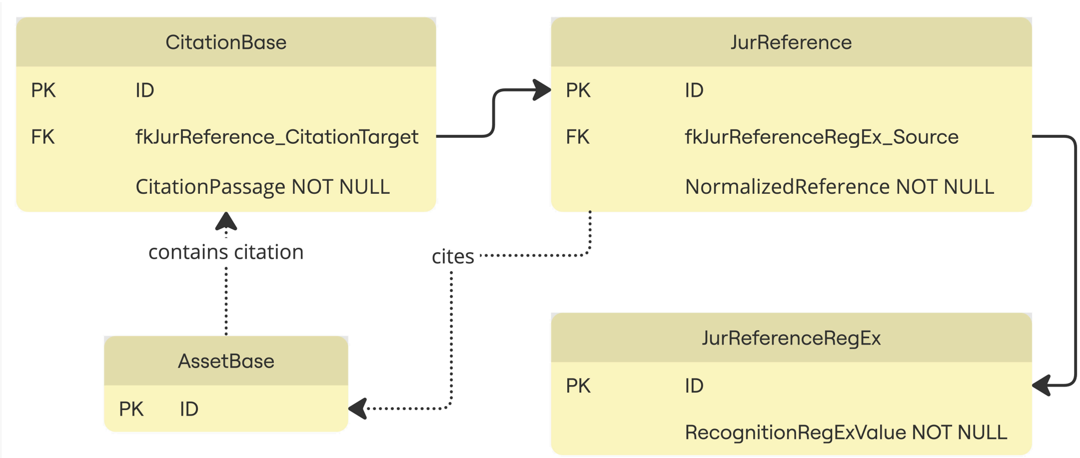



Figure 7: Database schema relevant for current solution of citation recognition

### 3.1.5 Challenges of the Regex-Based Approach

Using regular expressions for recognizing citations presents several challenges. General limitations of rule based methods for NER tasks are described in more detail in chapter 4.1.1. In the case of Swisslex, the following issues are particularly significant:

**Regex Maintenance** The regexes are difficult to maintain. Adding new regexes or updating existing ones requires a lot of manual work. The Swisslex employees managing the documents cannot manage the regexes themselves. Therefore, change requests mean a lot of work for Swisslex's IT Team. Different factors complicate maintenance:

- Equivalence and Redundancy: It is difficult to identify equivalent or redundant regexes.
- Location: The regexes are stored in different places (database and code).
- Dynamic variables: The expressions may be nested or contain dynamic variables.
- Finding regexes: There is no filter or search system available to manage the regexes.
- Testing: There is no automated testing system to verify the regexes.

**Special Cases** Several special cases make recognition difficult. Referencing multiple articles in a single citation poses a challenge for regular expression matching. Another issue is the ambiguity of abbreviations. For example, `CO` refers both to the Swiss Code of Obligations and can be used to anonymise people. It can be impossible to distinguish these cases with regexes alone. The special cases are described in more detail in chapter 3.3.4.

**Variants of Citation Styles** Different types of documents — such as judgments, laws, and journal articles — follow distinct citation rules, as described in chapter 3.2. Consequently, a specific rule must be defined for each citation style. Additionally, even though there are defined citation rules, not all citations follow them strictly. For example, some might leave out an obligatory part of the citation, such as the publication date.

**Generalization** Regexes cannot automatically generalize to unseen data and require manual maintenance. When a new law is introduced, regexes must be manually updated, otherwise the system will not recognize citations for the new law.

## 3.2 Citation Rules

The focus of this thesis lies on the recognition of citations of judgments, laws, and journals. Different citation styles are briefly described in the following chapters.

The rules described here are based on the document *"Zitierregeln"* from the Federal Supreme Court [5].

### 3.2.1 Judgments

Judgments of the Federal Supreme Court are cited differently depending on how they were published. There are three types, which will be addressed in the following subchapters.

#### 3.2.1.1 Officially Published Judgments of the Federal Supreme Court

The officially published judgments have been published in printed booklet form since 1875 [6]. Depending on how precisely one wants to specify certain parts of the judgment, the consideration or the page number can be added to the reference. In Table 7 some examples are shown.

A citation reference from the Federal Supreme Court consists of the abbreviation BGE, followed by the volume number, the subvolume number, the part number, the consideration and the page number [6].

- BGE 133 II 292 E. 3.2 S. 29

| Citation | Explanation |
|---|---|
| BGE 133 II 292 | Basic citation |
| BGE 133 II 292 E. 3.2 | Citation with consideration |
| BGE 133 II 292 E. 3.2 S. 29 | Citation with consideration and page number |
| BGE 132 III 18 E. 4.1; 132 V 321 E. 6.1 | Multiple cited judgments from different parts of the same volume, separated by a semicolon |
| BGE 128 IV 225 E. 2.3 S. 229, 232 E. 2 | Multiple cited judgments from the same volume and part, separated by a comma |

Table 7: Citation examples of officially published judgments (BGE)

There are special citation guidelines for old publications (up to volume 128) and hints (*"Hinweise"*), which are not discussed in detail.

#### 3.2.1.2 Unpublished Judgments of the Federal Supreme Court

Judgments that are neither published in the official collection nor reproduced in a legal journal are cited by the name of the court that issued the judgment (optional), the dossier number, the date of the judgment and if necessary the consideration. Some examples are given in Table 8.

These judgments are not printed, but available online.

| Citation | Explanation |
|---|---|
| Urteil (des Bundesgerichts) 5C.260/2006 vom 30. März 2007 | Basic citation |
| Urteil (des Bundesgerichts) 6B_214/2007 vom 13. November 2007 E. 5.10.3 | Citation with consideration |

Table 8: Citation examples of unpublished judgments (BGE)

#### 3.2.1.3 Judgments Published in Journals

For judgments published in a journal, the journal reference is given after a comma. This is introduced by *"in:"*. Some citations start with the journal reference first, followed by the dossier number, separated by a comma. If the consideration is to be included, it must be inserted before the journal or after the dossier number. The expressions in brackets are optional. Table 9 shows some examples.

| Citation | Explanation |
|---|---|
| • Urteil (des Bundesgerichts) 1P.440/2000 vom 1. Februar 2001, in: SJ 2001 I S. 221<br>• SJ 2001 I S. 221, 1P.440/2000 | Basic citation |
| • Urteil (des Bundesgerichts) 1P.440/2000 vom 1. Februar 2001 E. 2, in: SJ 2001 I S. 221<br>• SJ 2001 I S. 221, 1P.440/2000 E. 2 | Citation with consideration |

Table 9: Citation examples of judgments published in journals (BGE)

### 3.2.2 Other Federal Court Judgments

Judgments of the Federal Administrative Court are cited as described in Table 10.

| Citation | Explanation |
|---|---|
| BVGE 2007/2 | Basic citation |
| BVGE 2007/2 E. 3.2 S. 18 | Citation with consideration and page number |
| BVGE 2007/2 E. 3.2 S. 18 ; 2007/3 E. 3.3 S. 23 | Multiple cited judgments separated by a semicolon |

Table 10: Citation examples of the Federal Administrative Court

The Federal Criminal Court is cited as described in Table 11.

| Citation | Explanation |
|---|---|
| TPF 2005 142 | Basic citation |
| TPF 2005 127 E. 10.3.3 S. 134 | Citation with consideration and page number |
| TPF 2005 84 E. 3.2.1, 109 E. 5.2; 2004 34 E. 4.2 | Multiple cited judgments separated by a semicolon |

Table 11: Citation examples of the Federal Criminal Court

### 3.2.3 Other Courts (Cantonal, District, etc.)

In addition to the federal courts, there are also cantonal and district courts. The citation method differs depending on the court. Usually, they are cited by dossier number.

For example, a citation reference of the Social Insurance Court of the Canton of Zurich (Sozialversicherungsgericht des Kantons Zürich) would look like this: `AK.2001.00065`

Due to their large number, they are not discussed in detail.

### 3.2.4 Laws and Article of Laws

The following sections describe how laws and articles of laws should be cited in legal documents.

#### 3.2.4.1 Laws

In principle, the following elements should be listed when citing a law for the first time:

- Form of decree
- Date
- Full title or short title
- Abbreviation and reference (SR number) in brackets, separated by a semicolon

The following examples illustrate the citation of laws:
- Bundesgesetz vom 14. Dezember 1990 über die direkte Bundessteuer (DBG; SR 642.11)
- Waffenverordnung vom 21. September 1998 (WV; SR 514.541)

If laws are well-known across all areas, they can be cited in short form without further detail. For example: BV, ZGB, OR, StGB or AHVG.

For laws that do not have an official short title/abbreviation but are nevertheless frequently cited, one can create their own short forms. These must be specially labeled. The abbreviated form created must be indicated in brackets after the SR number with the introduction *"nachfolgend:"* or *"im Folgenden:"*. For example:

- Verordnung vom 2. Oktober 2000 über Massnahmen gegenüber Personen und Organisationen mit Verbindungen zu Usama bin Laden, der Gruppierung "Al-Qaïda" oder den Taliban (SR 946.203; nachfolgend: Talibanverordnung)
- Verordnung vom 24. Januar 1996 über die Unfallversicherung von arbeitslosen Personen (SR 837.171; im Folgenden: UVAL)

#### 3.2.4.2 Article of Laws

A handout from the university of Zurich [7] describes how articles of law should be cited. Those are to be cited with the following parts:

- Article ('Art.')
- Paragraph ('Abs.')
- Sentence ('Satz')
- Half-sentence ('Halbs.')
- Number ('Ziff.')
- Letter ('lit.')
- Abbreviation of the law (e.g. 'OR').

A citation must always be as precise as possible, i.e. the smallest applicable unit must be cited. If a letter belongs to the article number, it follows the number without a space (e.g. *Art. 321a CO*). If, on the other hand, the law uses letters to structure an article, there must be a space between the 'lit.' and the letter quoted. In the case of provisions that are no longer in force, the abbreviation of the law is preceded by a small *'a'* (stands for old) without a space (e.g. *aOR*). [7]

For example:

- Art. 335d Ziff. 1 OR
- Art. 343 Abs. 2 Halbs. 1 aOR
- Art. 10 Abs. 3 Satz 1 GlG
- Art. 336 Abs. 1 lit. d OR

### 3.2.5 Journals

Journals should be cited using the author's name, work title and journal reference, but often only the journal reference is stated. The journal reference contains:

- abbreviation of the journal
- volume number
- year
- subvolume number
- page number.

For correct citation, there is a list of journals and their citation methods in the document *"Zitierregeln"* from the Federal Supreme Court in chapter "Anhang 2" [5].

- MARKUS REICH, Das Leistungsfähigkeitsprinzip im Einkommenssteuerrecht, ASA 53 S. 16 f.
- JEAN-CLAUDE DE HALLER, L'hypothèque légale de l'entrepreneur, ZSR 101/1982 II S. 224

For lesser-known journals, the full name must be written the first time the journal is cited and the abbreviation must be placed in brackets:

- FABIEN GASSER, La participation à une organisation criminelle active dans le trafic de stupéfiants, Freiburger Zeitschrift für Rechtsprechung (FZR) 2006 S. 120 ff.

If the same article is cited more than once, the short form can be used:

- REICH, a.a.O., S. 16 f.
- DE HALLER, a.a.O., S. 225

## 3.3 Data Exploration

In this section, the relevant Swisslex data is explored and explained. It focuses on the distribution of the citations and documents in the database. The evaluation has been done between September and December 2024 and is based on the Swisslex INT database. This database contains a copy of the live Swisslex database, but might not be completely up-to-date. However, the general distributions should be comparable to the live database.

### 3.3.1 Documents

Swisslex contains a vast amount of documents. Each document has a status, which indicates whether it is e.g. published or not. There are approximately 950′000 documents with the status "published". The average length of the published documents is 72′664 characters. The following sections provide an overview of the distribution of languages, document types, and the amount of documents added over the last years.

#### 3.3.1.1 Languages

The majority of documents in the Swisslex INT database are in German (66%), followed by a significant amount of documents in French (29%). Other languages include Italian, English, and Romansh, but they are less frequent. For the analysis of the languages, only documents with the status "published" are considered. The distribution is shown in Figure 8.

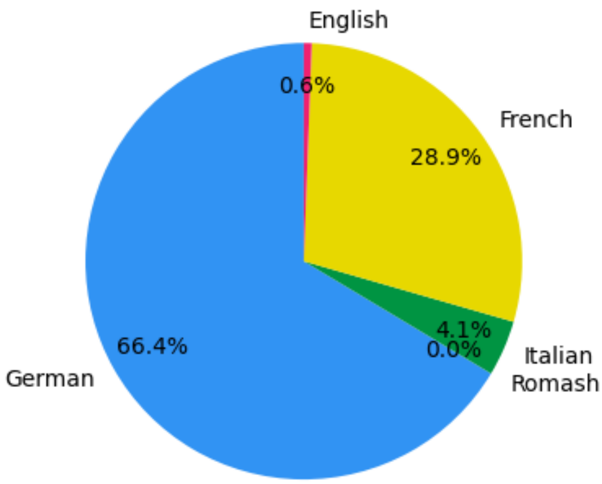



Figure 8: Language distribution

#### 3.3.1.2 Document Types

Most of the documents are case laws (77%). This term is in Swisslex used interchangeably with judgments. Hence, this thesis will also use the term "case law" for these documents. Further, there are also books, essays, commentaries, case law reviews and federal documents. The distribution of these types can be seen in Figure 9. The values are taken from the Swisslex website [4] on the 25.11.2024.

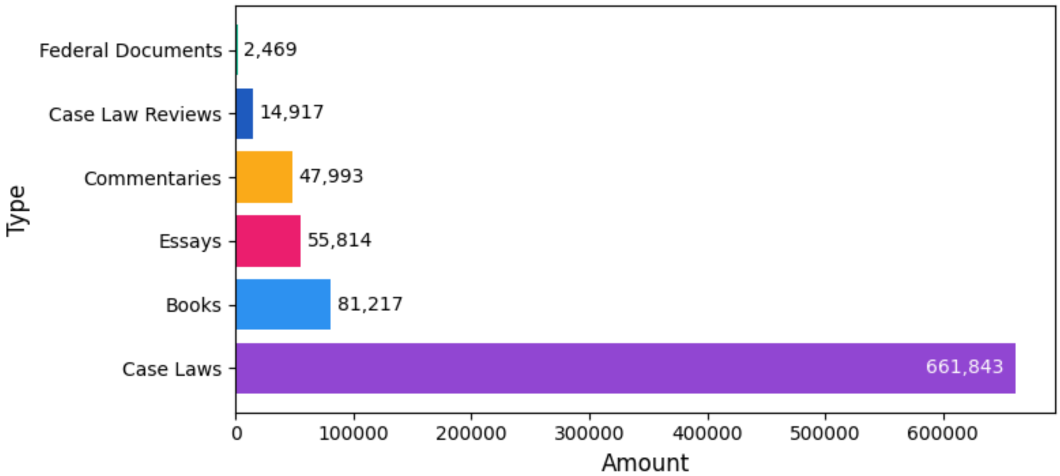



Figure 9: Swisslex document type distribution

#### 3.3.1.3 Cited Documents

The documents in Swisslex cite other documents. Most of the cited documents are either laws or case laws. Together, they make up 98% of the cited documents. The distribution of the cited documents is shown in Figure 10.

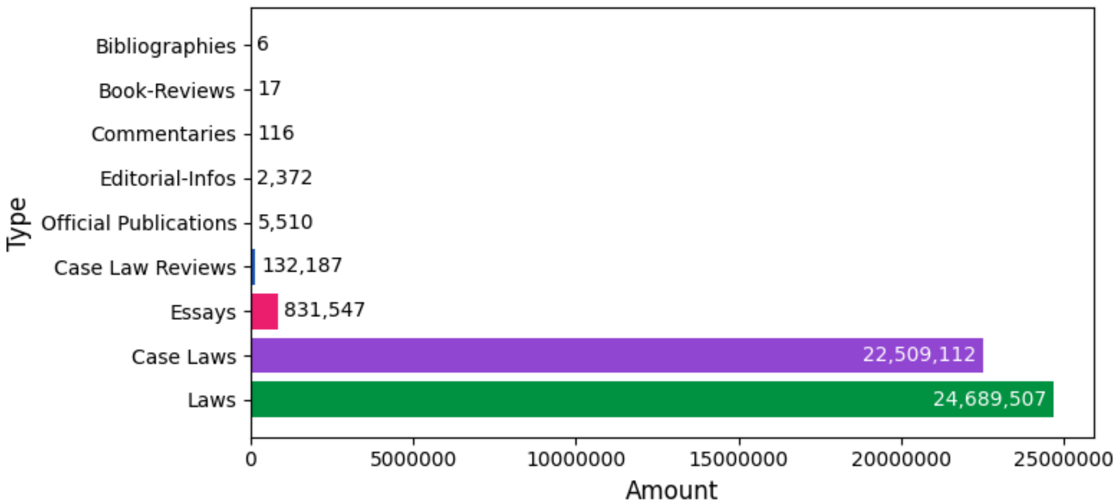



Figure 10: Swisslex cited documents distribution

#### 3.3.1.4 Amount of Documents per Year

The biggest part of the documents is from recent years. There are a few outliers, the oldest one being from 1835. To get an overview of the distribution of documents over the years, Figure 11 shows the amount of documents per year on a logarithmic scale.

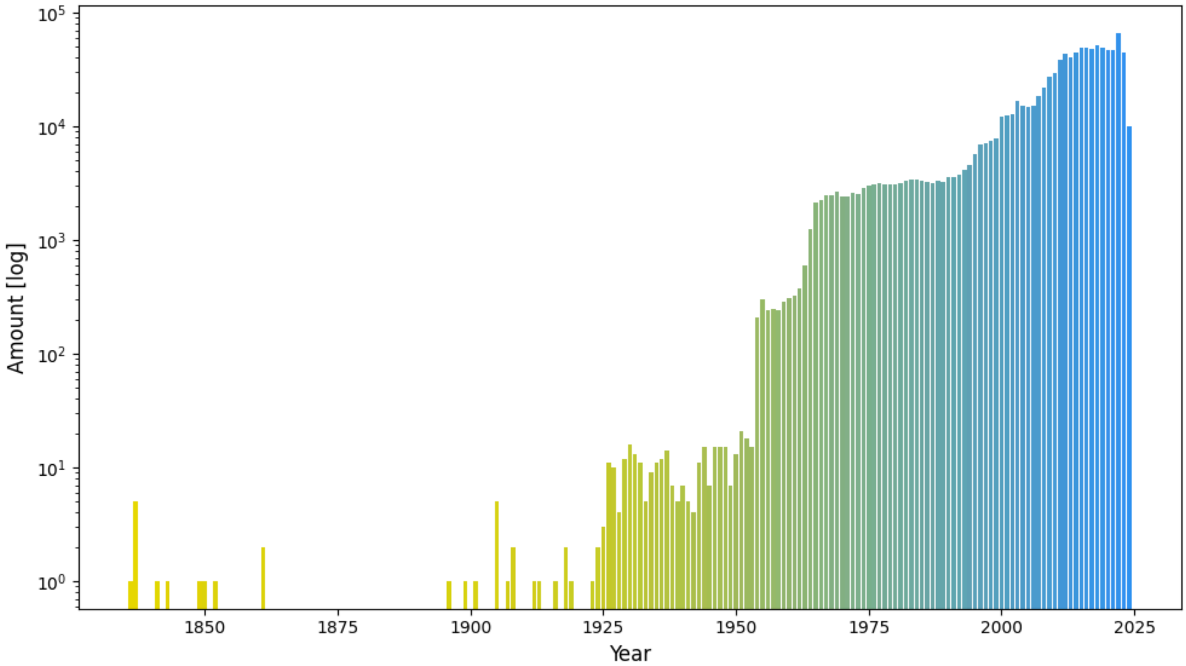



Figure 11: Swisslex documents per year (logarithmic scale)

### 3.3.2 Citations

In the following sections, the citations stored in the Swisslex database are analyzed more closely.

#### 3.3.2.1 Linking to Documents

Although over 60 million citations are registered in the database, only two-thirds link to documents available in Swisslex. A Swisslex employee suggests that citations not linked to documents are more likely misrecognized than missing from the database. However, due to the large number of citations, it is not feasible to verify this assumption.

#### 3.3.2.2 Length Distribution

Most citations are 1-20 characters long. There are a few outliers with up to 710 characters. The distribution of the citation lengths on a logarithmic scale can be seen in Figure 12.

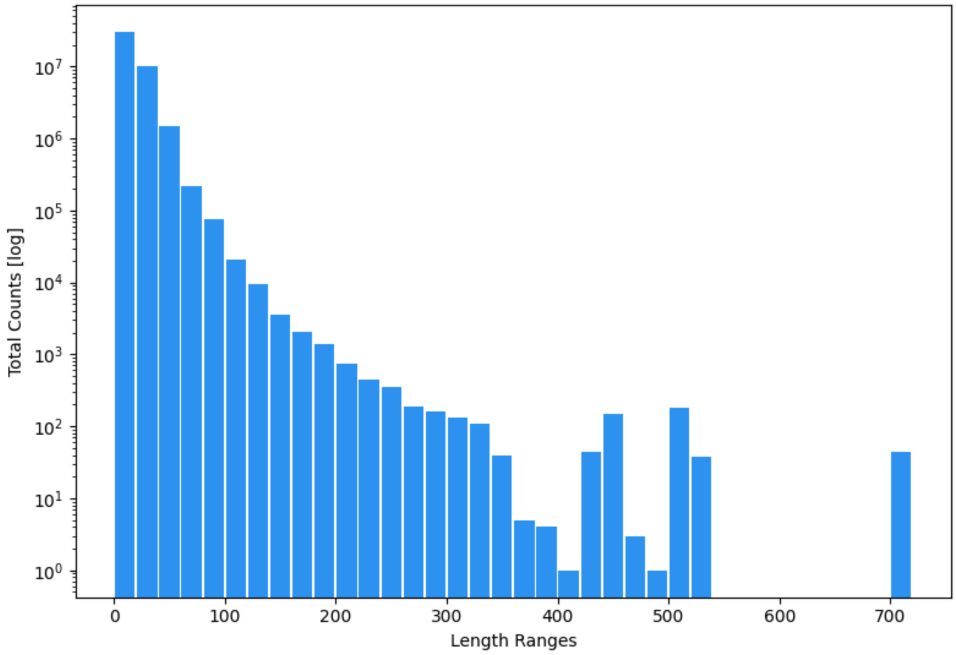



Figure 12: Citation length distribution (logarithmic scale)

#### 3.3.2.3 Duplicate Citations

There are many duplicate citation passages in the database. These are mainly citations of general laws. The most duplicates are for the citation passages "ZPO" and "OR", each with more than 300,000 duplicates. These are followed by "ZGB", "BGG", "LTF", "StGB", "StPO", "BV" and "VwVG", each of which was recognized as a citation between 100′000 and 200′000 times. The most cited specific judgment is "BGE 141 V 281". This passage is saved in the database more than 90′000 times. The most cited article is "Art. 42 BGG". This passage is cited about 50′000 times.

#### 3.3.2.4 Example of Citation Variants

A document can be cited correctly in different ways. For example, the judgment with the title *"33. Urteil vom 27. Juli 1973 i.S. G. gegen Ausgleichskasse des Kantons Luzern und Versicherungsgericht des Kantons Luzern"* from the Swiss Federeal Court has the citation reference `BGE 99 V 98`.

This document gets cited around 1′200 times by other documents in Swisslex. Among these citations, this judgment was cited in 72 different ways. Some variations occur because of different languages, others because of different ways to separate multiple citations.

Some examples include:

- `ATF 121 V 362 consid. 1b ; 99 V 98 consid. 4`
- `BGE 121 V 362 E. 1b; 99 V 98`
- `117 V 287 cons. 4, 99 V 98 cons. 4`
- `121 V 362 Erw. 1b und 99 V 98 Erw. 4`
- `ATF 99 V 98`
- `BGE 99 V 98`
- `DTF 99 V 98`
- `99 V 98`

The abbreviations ATF, DTF and BGE are the abbreviations in the respective national language for the decisions of the Federal Supreme Court [6]:

- ATF: Arrêts du Tribunal Fédéral Suisse
- DTF: Decisioni del Tribunale Federale Svizzero
- BGE: Entscheidungen des Schweizerischen Bundesgerichts

### 3.3.3 Parts of a Citation

Currently, there are no parts of the citation stored in the database. The only possibility to get the parts of a citation is to reverse-engineer the regular expression that was used to recognize the citation. This regex is stored in the database, as described in chapter 3.1.4.3, and sometimes contains named groups that represent the parts of the citation. However, these are often very generic and not complete.

As an example, the regular expression that recognizes a citation of the form

```
BGE 142 I 135 E. 1.6 S. 144
```

looks like this:

```
(?<PUB>[$Pub][[\\s,;()\\[\\]]{1,3}])(?<VOL:chaining([[  ]?[;,]  ])>[[1-9][0-9]{1,2}])
([[/\\-—?]][[1][9][0-9]{2}|[2][0][0-9]{2}])?[[\\s-–,;]{1,3}](?<PART>[I])[[\\s-–,;]
{1,3}][[1-9][0-9]{0,3}Extended_WithCatch]([[\\s,;()\\[\\]]{1,3}][(((del|du|vom)[[\\s]
{1,3}])|(dell'))[(((0)?[1-9]{1})|1[0-9]{1}|2[0-9]{1}|3[0-1]{1})][[\\s.]{1,2}][dec\\.]
[[\\s]{1,3}][[1][9][0-9]{2}|[2][0][0-9]{2}]]([[\\s,;()\\[\\]]{1,3}])?)?
```

The first named group `PUB` should capture `BGE` as publication, the second named group `VOL` should match `142` as volume, and the named group `PART` should capture `I` as part. However, the remaining parts of the ctiation, such as `135` as subvolume, `1.6` as consideration and `144` as page, are not captured in named groups.

The capturing groups are also named very generically, for example as `part A`. This makes it hard to understand which part of the citation should be captured by this group.

Apart from the capturing groups issues, it is difficult to apply the regular expression to a citation again to get the parts. This is due to the following reasons:

- Some regexes are not stored in the database but dynamically generated in the *Production Tool* code. Without the *Production Tool*, it is impossible to apply the regex to a citation.
- The regexes sometimes contain variables that are not resolved. This makes it impossible to apply them to a citation again, since they are invalid.
- Although most of the citations have been recognized by a regular expression, the regex is not always stored together with the citation. For most of the law citations, the matching regex is not stored in the database.

For some citations, individual parts are stored in the database. However, these represent only a small subset of the overall dataset and are both incomplete and error-prone. For instance, the `ArticleOfLaw` table contains parts of law citations but holds only 185,000 entries, while there are a total of 35.5 million law citations. Moreover, the table does not include the start and end indices for these components within the text.

For example, in the citation `Art. 3 Abs. 3 ZGB`, the table might include an entry for an `article` with the value `3`. However, it is unclear whether this `3` corresponds to the substring from index 5 to 6 or from index 12 to 13, as both positions contain the number 3. Accurate start and end indices are essential for fine-tuning a model, as they help define the exact spans of entities.

Moreover, the table is error-prone, with some parts being mislabeled. For instance, in the citation `Art. 138 Ziff. 1 Abs. 1`, the number 138 is identified as a `number` instead of an `article`. Such inconsistencies further reduce the reliability of the table as a source for accurate annotation.

Given the task outlined in chapter 1.3, it is essential to identify the parts of a citation as accurately as possible. This level of precision is crucial for linking the referenced document in a second step. Therefore, the existing data is insufficient for generating high-quality training data to fine-tune a model for part recognition.

### 3.3.4 Special Cases

This chapter describes some special cases of citations that are difficult to recognize with the current regex-based solution. The following examples are taken from the Swisslex database.

#### 3.3.4.1 Multiple Citations

The following Figure 13 shows an example of multiple citations. It is a screenshot from the judgment BGE 148 V 397 on swisslex.ch. It first refers to four different articles and paragraphs of the same ATSG. Immediately after this, a more complex multiple citation that was not correctly recognized by the existing system can be seen, which should refer to a large number of paragraphs of the IVG [8].

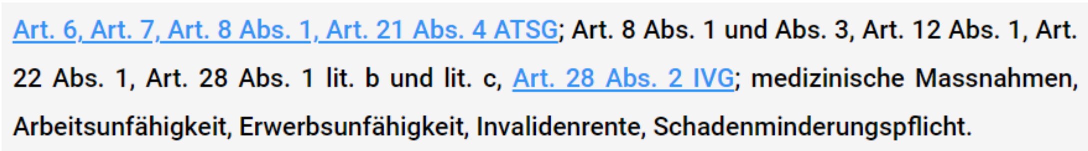

**Art. 6, Art. 7, Art. 8 Abs. 1, Art. 21 Abs. 4 ATSG; Art. 8 Abs. 1 und Abs. 3, Art. 12 Abs. 1, Art. 22 Abs. 1, Art. 28 Abs. 1 lit. b und lit. c, Art. 28 Abs. 2 IVG; medizinische Massnahmen, Arbeitsunfähigkeit, Erwerbsunfähigkeit, Invalidenrente, Schadenminderungspflicht.**

Figure 13: Example of a failed recognition of multiple citations

#### 3.3.4.2 Amibguity of Abbreviations

Another special case is the ambiguity of abbreviations. For example, various Italian-speaking courts anonymise persons with CO, marked as CO 1 in Figure 14. At the same time, CO is also the French and Italian abbreviation for the Swiss Code of Obligations. In particular, if only the OR/CO as a whole is referred to and there is only one anonymised person (the numbering is therefore omitted for anonymisation purposes), it is often not clear without the context whether a person or a law is meant. [8]

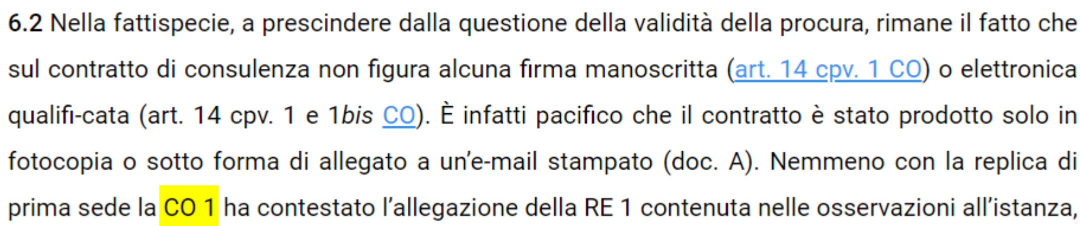

**6.2** Nella fattispecie, a prescindere dalla questione della validità della procura, rimane il fatto che sul contratto di consulenza non figura alcuna firma manoscritta (art. 14 cpv. 1 CO) o elettronica qualifi-cata (art. 14 cpv. 1 e 1*bis* CO). È infatti pacifico che il contratto è stato prodotto solo in fotocopia o sotto forma di allegato a un'e-mail stampato (doc. A). Nemmeno con la replica di prima sede la CO 1 ha contestato l'allegazione della RE 1 contenuta nelle osservazioni all'istanza,

Figure 14: Example of a ambigious abbreviations

#### 3.3.4.3 Cantonal Law Misinterpreted as Federal Law

Some laws exists in more than one canton. To distinguish between the different cantons, the cantons abbreviation is usually added to the law. For example, the law "Gesetz über die Verwaltungsrechtspflege" (VRPG) exists in several cantons which look like this:

- VRPG/AG: Aargau
- VRPG/BE: Bern

The abbreviation VRPG alone is not unique and can refer to different cantonal laws. The current system does not take this into account, misses the cantonal abbreviation and thus interprets the wrong law.

#### 3.3.4.4 Misinterpreting Systematic Collections of Law as Laws

Systematic collections of law from a canton (e.g. BSG = Bernische Systematische Gesetzessammlung) include all laws of a canton. The current system interprets these systematic collections sometimes as laws, which is incorrect.

For example, SG 291.100 is the “Advokaturgesetz” from canton Basel-Stadt [9]. The current system recognizes only “SG 291” in form of article 291 from the law SG.

#### 3.3.4.5 Deviations from the Standard Citation Format

The standard citation format is not always followed. Spelling mistakes and incorrectly formatted citations may occur. For example, the order of the citation elements may be different. Instead of `Art. 3 ZGB`, the citation might be written as `ZGB Art. 3`.

### 3.3.5 Data Duplication

There are two cases of data duplication in the documents.

Firstly, courts may publish updated versions of their judgments. In such cases, the judgment is resent and stored as a new document in Swisslex. A status value is assigned to each document, with 700 indicating that the judgment is “published”. Consequently, a single judgment may correspond to several Swisslex documents, but only the one with a status of 700 represents the definitive and most recent version.

Secondly, judgments may also appear redundant when published by third parties, such as legal journals. In these cases, the same judgment is recorded twice in the dataset: once as printed in the journal and once directly from the court. In this case, both documents have a status of 700, as they are both published.

# 4 Theoretical Foundations

This chapter outlines the theoretical foundations for solving the problem of citation recognition. It also highlights related work conducted in the field of NER.

## 4.1 Approaches to Named Entity Recognintion

Citation recognition is a subtask of NER [10]. Recognizing named entities is a well-known problem in the field of natural language processing, especially when it comes to identifying common types of named entities, such as people, locations, and organizations.

In the legal domain, the complexity of legal texts and the limited availability of training data, especially within the Swiss legal framework, present significant challenges. There are some studies that have tackled these problems before. However, most of them focus on the US or European legal system or use outdated technologies.

The following sections discuss and compare different approaches to NER and their application to citation recognition.

### 4.1.1 Rule-Based Methods

Rule-based methods rely on manually crafted rules to identify and classify named entities. This approach can be effective in domains with well-defined entity categories, like legal or medical texts, where specific terms or phrases are predictable. However, rule-based methods have several limitations in scalability and adaptability [11]. The next sections describe different rule-based methods used for NER.

#### 4.1.1.1 Regular Expressions (Regex)

Regular expressions (regex) are patterns used to define the format of text and match entities. For instance, a regex like `d{2}/d{2}/d{4}` could match dates in the DD/MM/YYYY format. The current Swisslex solution uses a complex set of regular expressions to identify citations in legal documents. These regexes are based on patterns commonly found in legal citations, such as the use of specific abbreviations, punctuation, and formatting conventions. While this approach can be effective in identifying citations in a controlled environment, it lacks the flexibility and generalization capabilities of context- and sequence-based models. [12]

In a paper published in 2005, a rule-based system for extracting references from legal text was proposed, similar to the current regex-based approach in Swisslex. [13]

#### 4.1.1.2 Linguistic Patterns

Linguistic patterns are syntactic rules that rely on sentence structure, such as using part-of-speech tagging to identify names following titles like “Judge” or “Mr.” On the other hand, contextual rules consider the surrounding text of potential entities. For instance, legal cases can be identified by detecting terms like “v.” or “versus” between names. [14]

In 2019, a paper from French researchers describes a system for extracting legal citations using a rule-based approach with the Java Annotation Patterns Engine. [15]

#### 4.1.1.3 Vocabulary Lists (Gazetteers)

Vocabulary lists are predefined lists of known entities or terms, which are matched against the text to identify occurrences. For example, a gazetteer containing known legal case names can be used to automatically detect and extract references to those cases in documents. [16]

#### 4.1.1.4 Advantages and Limitations

Following are some advantages and limitations of rule-based methods for NER.

**Advantages**
- High precision in controlled environments
- Effective for well-defined, static domains with limited variability
- Do not require large amounts of labeled data for training

**Limitations**
- Low scalability to large and diverse datasets
- High maintenance cost as rules need to be updated frequently
- Limited intelligence since rules are handcrafted through pre-defined patterns and heuristics and may not generalize to small variations in text

### 4.1.2 Statistical Methods

Shifting away from manual rules, statistical methods such as Hidden Markov Models (HMM) and Conditional Random Fields (CRF) use probabilistic models. They identify named entities based on patterns learned from annotated datasets. These approaches are well-suited for applications with abundant labeled data. They excel at handling diverse textual content but are highly dependent on the quality and comprehensiveness of the training data. [10]

One of the earliest known works explicitly defining NER in the legal domain is described by Dozier et al. [17]. The authors applied NER to U.S. case law, depositions, pleadings, and other legal documents using a combination of simple list-based lookups, contextual rules, and statistical models. They developed specialized taggers for identifying jurisdictions, courts, titles, document types (e.g., briefs, memoranda), and judges. The jurisdiction tagger outperformed the others, achieving an F1 score of 92, while the remaining taggers attained F1 scores ranging from 82 to 85.

#### 4.1.2.1 Hidden Markov Models (HMM)

Hidden Markov Models (HMMs) are generative statistical models often used in NER tasks. They are designed to handle sequences of data, where observable events (like words in a sentence) depend on underlying hidden states (like entity labels). As generative models, HMMs calculate the joint probability of both the observed data and the hidden states. This allows them to generate new data based on learned patterns. They rely on the Markov assumption, which means the current hidden state depends only on the previous one. Also, they model the likelihood of observable events being emitted from these states.

The core strength of HMMs lies in their ability to capture temporal dependencies between entities. This makes them effective for sequential data like language. However, their performance is highly dependent on the availability of well-annotated training data and predefined probability distributions. HMMs are computationally efficient and have been successfully applied to tasks such as part-of-speech tagging and speech recognition, yet they may struggle with complex feature dependencies. [12], [18]

#### 4.1.2.2 Conditional Random Fields (CRF)

Conditional Random Fields (CRFs) are a discriminative probabilistic model used for sequence labeling tasks like NER. CRFs model the conditional probability of a sequence of labels given the input data, allowing them to capture dependencies between adjacent labels. This makes them particularly effective for tasks where the label of a word is influenced by its surrounding context. CRFs have been widely used in NER tasks due to their ability to model complex interactions between features and labels. [12]

They are especially effective when combined with deep learning models like Bidirectional LSTMs, enabling the model to capture both past and future context in sequences. However, CRFs require manual feature engineering and are computationally intensive. This makes them less flexible than deep learning models in handling large-scale datasets. This is discussed in more detail in chapter 4.1.4.1.

#### 4.1.2.3 Advantages and Limitations

Here are some advantages and limitations of statistical methods for NER tasks.

**Advantages**

- Effective in capturing temporal dependencies and contextual information
- Suitable for tasks requiring sequence labeling and structured output
- Can model complex interactions between features and labels

**Limitations**

- Require manual feature engineering
- Highly dependent on the quality and quantity of annotated training data
- Computationally intensive, CRFs struggle with large datasets

### 4.1.3 Machine Learning Methods

In contrast to the rule-based approach, machine learning classifiers can be used for NER tasks in a more flexible way.

Such classifiers are trained in a supervised manner. Being provided with a set of labeled data, they learn the patterns in the data and can then predict the labels of new data points. This allows them to recognize unseen patterns and generalize from the training data. Common classifiers include Support Vector Machines (SVM), Random Forest, Decision Trees, K-Nearest Neighbors (KNN), and others. [19]

Using machine learning classifiers for NER has been a common practice for many years. In 2001, Yamada et al. showed that Support Vector Machines were effective for NER in Japanese, achieving an F-value of 83. [20]

For citation recognition in legal documents, classifiers must precisely identify parts of a citation, such as paragraphs or sections. This requires a multi-class classifier and sufficient labeled data for training.

Machine learning classifiers have limitations. They cannot recognize new entity types that are not present in the training data. They also require a large labeled corpus for effective training, which can be computationally expensive. Additionally, they are less flexible when encountering small errors in the training data, such as typographical errors. [21]

#### 4.1.3.1 Advantages and Limitations

In the context of NER, machine learning methods have the following advantages and limitations.

**Advantages**

- Can recognize unseen patterns and generalize from the training data, more flexible than rule-based approaches
- Adaptable to different domains and languages
- Maintenance is less expensive

**Limitations**

- Cannot recognize new types or classes that were not present in the training data
- Requires a large amount of labeled data for training, computationally expensive
- Less flexible in handling small errors in the training data

### 4.1.4 Deep Learning Methods

Deep learning methods have revolutionized the field of NER by enabling models to learn complex patterns and representations directly from data. These methods leverage neural networks to automatically extract features and capture dependencies in sequences. This makes them highly effective for NER tasks. Deep learning models have achieved state-of-the-art performance on various NLP tasks, including NER.

#### 4.1.4.1 RNNs and LSTMs

Recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) are designed for handling sequence prediction tasks. RNNs are capable of processing sequential data by maintaining a form of memory across inputs. LSTMs, a specific variant of RNNs, excel at capturing long-term dependencies in sequences. This allows them to effectively retain information over extended periods, which is crucial for understanding context and detecting entities in texts. [22]

However, RNNs and LSTMs have limitations in capturing bidirectional context. They process sequences in a forward or backward direction. This can lead to suboptimal performance in tasks where the context of a word is influenced by both preceding and following words.

The combination of Bidirectional LSTMs with CRFs (BiLSTM-CRF) became a widely used architecture for NER. It enables the model to capture both past and future context in sequences. The BiLSTM component processes the input sequence in both directions, capturing information from past and future tokens. The CRF layer then models the dependencies between labels in the sequence. This allows the model to predict the most likely sequence of labels for named entities. This architecture has shown significant improvements in NER tasks. It reduces the need for manual feature engineering and achieves state-of-the-art performance on benchmark datasets. [12], [23]

Leitner et al. focused on fine-grained NER in legal documents for the German legal framework. In a study on NER in German legal texts, the authors compiled a dataset of 67,000 sentences from court decisions, annotated with 54,000 entities spanning 19 fine-grained semantic classes (later grouped into 7 coarse-grained classes), including legal citations. They trained CRFs and BiLSTM networks, achieving the highest F1 scores with the BiLSTM models: 95.46 for the fine-grained classes and 95.95 for the coarse-grained classes. This research was conducted within the LYNX project, which aims to develop a semantic platform for advanced document processing and analysis in the legal domain. [24]

#### 4.1.4.2 Transformer-Based Models

In 2017, a team of researchers from Google presented the transformer architecture, which would revolutionize NLP research. It is an implementation of the encoder-decoder model. In the encoder, the input sequence is processed and transformed into a sequence of hidden states, which are then used by the decoder to generate the output.

However, the key innovation of the transformer architecture is the self-attention mechanism. This allows the model to focus on different parts of the input sequence simultaneously at each timestep. Different weights (“attention”) are assigned to input tokens based on their relevance to the current token, allowing the model to learn more complex patterns in the data. This stands in contrast to previous deep neural networks for NLP tasks, which were based on RNNs. Their sequential nature made it difficult to fully understand context in long sequences of text.

Following Google’s publication, various transformer-based models have been developed. The two best known are GPT and BERT, which have achieved excellent performance in NLP tasks. In the following sections, these models will be discussed in more detail. [25], [26]

### Generative Pre-Trained Transformer (GPT)

A generative pre-trained transformer (GPT) is a model that takes only the decoder part of the transformer architecture and uses it for language modelling. It belongs to the category of large language models (LLMs). The first GPT model was presented by OpenAI in 2018. Since then, several improved versions have been published by OpenAI and other research groups. [27]

These GPT models are trained using self-supervised learning, which is a subset of unsupervised learning. The general idea is that the model predicts the next token in a sequence and then calculates the loss based on the difference from the actual token. In this way, the model generates its own supervision signal from the data. This approach allows GPT models to be trained on large amounts of unlabeled data, which is available in abundance. The learning is unidirectional because it only looks at the previous context to predict the next word. By checking the loss at each step, the model learns to generate sequences similar to the input data. [26], [28]

GPT models are foundational models. They are trained on a broad range of diverse datasets to gain a general understanding of language. This step is computationally expensive. To use the model for a specific task, such as citation recognition, transfer learning is used. After pre-training, the model can be fine-tuned on a smaller set of labeled data to gain more specific knowledge about the task. This is often done using supervised learning or reinforcement learning. However, the model does not necessarily need to be fine-tuned. It can be used out of the box. [27], [28]

While GPT models have primarily been used for text generation tasks, it is also possible to use them for NER. However, models specifically designed for NER perform better. For this reason, additional fine-tuning on a dataset of named entities is required to achieve better results.

There are certain fine-tuned versions of GPT models available that can be used for NER tasks, such as GPT-NER. Proposed by researchers in 2023, this model addresses the challenge by reframing sequence labeling as a generation task, making it more adaptable for LLMs. [29]

### Bidirectional Encoder Representations from Transformers (BERT)

Another important model in the area of NLP is BERT, initially published by Google in 2018. This model is also based on the transformer architecture but uses only the encoder part. The main innovation of BERT is the introduction of masked language modeling. During training, 15% of the input tokens are randomly masked. The model is trained to predict the value of these masked tokens based on context. This allows BERT to learn bidirectional representations of the input text, since the model has to consider context both on the left and the right to make a prediction. The technique also uses unsupervised pre-training on large amounts of text data.

The second task that BERT is trained on is next sentence prediction. The model is presented with pairs of sentences and has to predict whether the second sentence follows the first one in the original text. This task helps the model learn relationships between sentences and improves its understanding of context.

Through this training process, BERT learns to understand language in general. However, the model does not come with an output layer. Instead, to use it for a specific task like NER, a new output layer is added. The model has to be fine-tuned on a task-specific dataset using supervised learning. [30], [31]

In recent years, the use of deep learning models, especially transformer-based models like BERT, has become increasingly popular in the field of NLP. For example, a German research group developed a *German BERT* model [32] for legal NER based on the dataset created by Leitner et al [24]. The recognized entities are divided into the following categories: `Person`, `Judge`, `Lawyer`, `Country`, `City`, `Street`, `Landscape`, `Organization`, `Company`, `Institution`, `Court`, `Brand`, `Law`, `Ordinance`, `European legal norm`, `Regulation`, `Contract`, `Court decision`, and `Legal literature`.

For the English language, researchers at the University of Athens have fine-tuned BERT-based models on legal text and published their resulting *Legal-BERT* models. They are not fine-tuned for NER. [33]

A Swiss researcher, Joel Niklaus, is working in the field of legal NLP. He collaborated on a legal benchmark called *LegalBench* [34], which aims to measure legal reasoning in LLMs. He has also led research projects for the Swiss Federal Supreme Court [35]. As part of his work, he pre-trained multiple BERT-based models on his own legal dataset. The dataset is called *Multi Legal Pile* [36], and covers 24 languages and multiple legal sources. The models can be found on the Hugging Face model hub and would have to be further fine-tuned on downstream tasks.

Another project focussing on Swiss texts is the *SwissBERT* [37], developed by researchers from the University of Zurich. This model is fine-tuned on Switzerland-related news articles in the languages German, French, Italian, and Romansh. To be used for NER, it would have to be further fine-tuned.

**GLiNER**

In 2023, a new model called GLiNER was proposed. The model builds on BERT, but is specifically designed for NER. While BERT assigns a label to each token, GLiNER aims to match entity types to spans (sequences of tokens). The model takes the entity types as input together with the text. It then processes both using a bidirectional transformer and computes matching scores between entity type embeddings and text spans.

This approach allows for a parallel extraction of multiple entity types, making GLiNER faster than BERT in handling NER tasks, especially if multiple entities are involved. The model has been shown to outperform larger models such as ChatGPT in NER, despite having a much smaller parameter size. [38]

#### 4.1.4.3 Advantages and Limitations

The specific advantages and limitations of deep learning models depend on the model architecture. However, in general, they share the following characteristics.

**Advantages**

- Effective in capturing long-term, bidirectional dependencies
- Achieve state-of-the-art performance in NER tasks
- Reduce the need for manual feature engineering

**Limitations**

- Computationally expensive
- Require large amounts of labeled data for training
- Require fine-tuning for specific tasks

## 4.2 Training Strategies for Transformer Models

There are many training strategies to make use of deep learning methods. The following sections will give an overview over relevant techniques and methods for the project.

### 4.2.1 Pre-Training

Pre-training refers to the initial phase of training a model on a large, general-purpose dataset to learn foundational patterns, representations, or features. For language models, this often involves unsupervised learning on vast text corpora such as Wikipedia [39] or other large-scale text datasets. The goal is to capture general linguistic patterns, syntax, semantics, and contextual relationships between words. Models like BERT exemplify this approach. Tasks such as masked language modeling enable the development of transferable representations. This process requires significant computing resources and large data sets, but provides a robust foundation for transfer learning. [30]

Some models undergo continued pre-training on domain-specific or task-specific datasets to further refine their representations [40]. In the legal domain, models like *joelniklaus/legal-xlm-roberta-large* [41] start from the standard XLM-RoBERTa model and are additionally pre-trained on the Multi Legal Pile dataset [42], a multilingual corpus of legal texts covering 24 languages. This additional pre-training is intended to enhance the model's ability to handle legal terminology and context, potentially improving its performance on downstream tasks in the legal field.

### 4.2.2 Zero-Shot and Few-Shot Prompting

In recent years, the performance of out-of-the-box, general-purpose LLMs has improved significantly on a variety of NLP tasks, like NER. It allows users to perform zero-shot and few-shot prompting. These approaches are particularly useful when the entity types are not known in advance, or when there is insufficient training data to fine-tune a model. [43]

In the following section, different approaches and examples using GPT-4o [44] are shown. While this model is not suitable for citation recognition in this project due to the NFRs, it offers valuable insight into the current state of the art performance of LLMs.

#### 4.2.2.1 Zero-Shot Prompting

Pre-trained transformer models like BERT, as well as LLMs such as GPT-4o or Llama 3, demonstrate remarkable zero-shot prompting capabilities. These models can perform NER on unseen entity types by leveraging knowledge gained during pre-training. This is particularly useful in scenarios where the entity types are not known in advance, or where the number of entity types is frequently changing. [43]

```
**Input text:** Am 1. Januar 2022 trat das revidierte Bundesgesetz vom 19. Juni
1959 über die Invalidenversicherung (IVG; SR 831.20) in Kraft (Weiterentwicklung
der IV [WEIV]; Änderung vom 19. Juni 2020; AS 2021 705; BBl 2017 2535). Die dem
angefochtenen Entscheid zugrunde liegende Verfügung der IV-Stelle erging vor dem 1.
Januar 2022. Nach den allgemeinen Grundsätzen des intertemporalen Rechts und des
zeitlich massgebenden Sachverhalts (BGE 144 V 210 E. 4.3.1; BGE 129 V 354 E. 1) sind
daher die Bestimmungen des IVG und diejenigen der Verordnung vom 17. Januar 1961 über
die Invalidenversicherung (IVV; SR 831.201) sowie des Bundesgesetzes vom 6. Oktober
2000 über den Allgemeinen Teil des Sozialversicherungsrechts (ATSG; SR 830.1) in der
bis 31. Dezember 2021 gültig gewesenen Fassung anwendbar (BGE 148 V 174 E. 4.1).
**Task:** Extract all legal citations from the input text. Each citation should
include references to laws, amendments, and specific articles or sections.
**Output:**
```

Code-Snippet 3: Zero-shot prompting example input

The output is then added to the *Output:* section of the prompt. The model can generate the desired output based on the input text and the provided task.

```
1. IVG; SR 831.20
2. Änderung vom 19. Juni 2020; AS 2021 705; BBl 2017 2535
3. BGE 144 V 210 E. 4.3.1
4. BGE 129 V 354 E. 1
5. IVV; SR 831.201
6. ATSG; SR 830.1
7. BGE 148 V 174 E. 4.1
```

Code-Snippet 4: Zero-shot prompting example output

#### 4.2.2.2 Few-Shot Prompting

In few-shot prompting, the model is provided with a small number of examples of the task it is supposed to perform. This allows the model to learn the task with minimal supervision. Few-shot prompting is useful when training data is scarce or when the model needs to be quickly adapted to a new task. [45]

```
## Example 1
**Input:** Gemäss Art. 64a Abs. 1 des Bundesgesetzes über die Alters- und Hinterlassenenversicherung (AHVG; SR 831.10) sowie Art. 10 Abs. 2 des Bundesgesetzes über die Unfallversicherung (UVG; SR 832.20) werden die Leistungen angepasst.

**Output:**
- Art. 64a Abs. 1 des Bundesgesetzes über die Alters- und Hinterlassenenversicherung (AHVG; SR 831.10)
- Art. 10 Abs. 2 des Bundesgesetzes über die Unfallversicherung (UVG; SR 832.20)

## Example 2
**Input:** Das Bundesgesetz über die Ergänzungsleistungen zur Alters-, Hinterlassenen- und Invalidenversicherung (ELG; SR 831.30) wurde zuletzt im Jahr 2020 (AS 2020 333) geändert.

**Output:**
- Bundesgesetz über die Ergänzungsleistungen zur Alters-, Hinterlassenen- und Invalidenversicherung (ELG; SR 831.30)
- AS 2020 333

## Prediction
**Input:** Am 1. Januar 2022 trat das revidierte Bundesgesetz vom 19. Juni 1959 über die Invalidenversicherung (IVG; SR 831.20) in Kraft (Weiterentwicklung der IV [WEIV]; Änderung vom 19. Juni 2020; AS 2021 705; BBl 2017 2535). Die dem angefochtenen Entscheid zugrunde liegende Verfügung der IV-Stelle erging vor dem 1. Januar 2022. Nach den allgemeinen Grundsätzen des intertemporalen Rechts und des zeitlich massgebenden Sachverhalts (BGE 144 V 210 E. 4.3.1; BGE 129 V 354 E. 1) sind daher die Bestimmungen des IVG und diejenigen der Verordnung vom 17. Januar 1961 über die Invalidenversicherung (IVV; SR 831.201) sowie des Bundesgesetzes vom 6. Oktober 2000 über den Allgemeinen Teil des Sozialversicherungsrechts (ATSG; SR 830.1) in der bis 31. Dezember 2021 gültig gewesenen Fassung anwendbar (BGE 148 V 174 E. 4.1).

**Output:**
```

Code-Snippet 5: Few-shot prompting example input

The model is then provided with the input text and the task it is supposed to perform. The output is generated based on the input text and the examples provided.

```
**Output:**
- Bundesgesetz vom 19. Juni 1959 über die Invalidenversicherung (IVG; SR 831.20)
- Weiterentwicklung der IV [WEIV]; Änderung vom 19. Juni 2020
- AS 2021 705
- BBl 2017 2535
- BGE 144 V 210 E. 4.3.1
- BGE 129 V 354 E. 1
- Verordnung vom 17. Januar 1961 über die Invalidenversicherung (IVV; SR 831.201)
- Bundesgesetz vom 6. Oktober 2000 über den Allgemeinen Teil des Sozialversicherungsrechts (ATSG; SR 830.1)
- BGE 148 V 174 E. 4.1
```

Code-Snippet 6: Few-shot prompting example output

#### 4.2.3 Fine-Tuning

After pre-training, fine-tuning is a crucial step in adapting models to specific tasks or domains. During fine-tuning, the final layers of the model, called the head, are adapted to task-specific datasets while retaining the general knowledge from earlier layers. For example, a BERT model with a token classification head can be fine-tuned on a legal dataset to effectively perform NER tasks. [26], [46]

In recent years, other models like Llama have also demonstrated their effectiveness in being fine-tuned on task-specific datasets, achieving performance gains that surpass zero-shot or few-shot prompting capabilities. [47]

The fine-tuning process typically involves the following steps:

1. **Data Preparation**: Collecting and preprocessing training data. This includes extracting text data from existing sources, labeling entities, splitting and shuffling the data, and converting it into a format compatible with the model.
2. **Model Selection**: Choosing a pre-trained transformer model, such as BERT, DistilBERT or Llama3, as the base model for fine-tuning.
3. **Fine-tuning**: Training the model on the labeled training data by updating its weights to minimize the task-specific loss function. This step often involves freezing some layers to retain general pre-trained features while adapting the task-specific layers.
4. **Evaluation**: Evaluating the model's performance on a separate validation set. This helps to tune hyperparameters and prevents overfitting.
5. **Inference**: Using the fine-tuned model to make predictions on new, unseen data using an API or UI.

The fine-tuning process can be repeated multiple times with different hyperparameters, architectures, and training data to optimize the model's performance.

#### 4.2.4 Knowledge Distillation

Knowledge distillation is the process of transferring knowledge from a complex teacher model to a simpler student model. Ideally, the smaller student model matches the performance of the larger teacher model while operating significantly faster. This approach enhances deployment inference speed and significantly lowers computational costs.

This principel was originally presented in 2006 under the term "Model Compression" [48]. The term "distillation" was introduced by Hinton et al. in 2015 [49].

Knowledge distillation is particularly relevant for LLMs due to their high computational demands. In response-based distillation, a few-shot Llama model, for example, can act as the teacher. It generates task-specific synthetic annotations for unlabeled samples. These outputs are then used to fine-tune a smaller BERT model as the student. The focus lies on the teacher's predictions rather than probability distributions. This method creates efficient, task-specific models that match the performance of larger LLMs while offering flexibility across different architectures. [50], [51], [52]

A Hugging Face blog article [53] highlights that fine-tuning smaller models on synthetic data also drastically reduces inference costs. For instance, a RoBERTa-base model processes 1 million sentences for 2.70 USD compared to 3061 USD with GPT-4. It achieves an F1 score of 94 — matching GPT-4—while providing faster inference at 0.13 seconds per sentence, unlike the several seconds required by GPT-4.

#### 4.2.5 Model Chaining

Model Chaining is a technique where multiple models are linked in a sequential process, each model specialized in a specific task. The output of one model is used as input for the next model. By breaking down tasks, model chaining can improve performance. This is possible as each model in the chain can focus on a distinct step, such as classifying entities before extracting finer details. [54]

In the context of citation recognition, model chaining could be applied by first using a model to identify the broad type of reference (e.g. "law" or "case law") and then using a subsequent model to extract specific components like article numbers or page references. For example, since a law citation contains different parts than a case law citation, it may be beneficial to use different models for each type of citation. Through this approach, the models can be optimized for their specific task. The training data for each model could be tailored to the respective citation type, leading to a more consistent dataset for training. Additionally, it would be possible to use different follow-up models depending on the type of citation identified in the first step.

Recent research has shown that model chaining can improve the performance of LLMs. A group of researchers have published a paper in November 2024, where they evaluated an LLM on legal information extraction tasks using a prompt chaining technique. They worked with a few-shot setting, where the model had to classify the information contained in the text. It then used this classification to select the relevant prompt for information extraction. With this method, the F1 score for entity extraction improved by approximately 4% compared to a baseline model. [55]

## 4.3 Hyperparameters

This chapter describes the importance of hyperparameters and specific values that can be optimized.

### 4.3.1 Overview

Hyperparameters control the learning process of a machine learning or deep learning model. Unlike model parameters, which are learned during training, hyperparameters are set before the training begins. They determine how the model behaves during the learning phase. Choosing the right hyperparameters can be challenging, as they are not automatically learned and must be selected manually. However, their proper selection can significantly impact the model's performance and speed. [22]

According to Goodfellow et al. [22], hyperparameters can be optimized in two ways:

- Manual hyperparameter tuning
- Automatic hyperparameter optimization algorithms

Manual hyperparameter tuning is time-consuming and inefficient. Thus, automatic hyperparameter optimization algorithms can be used to find the best hyperparameters for a given model. These algorithms are based on optimization techniques such as grid search, random search, or Bayesian optimization. The only downside is that they require a lot of computational resources and can be computationally expensive. [22], [56]

In the following chapter, the hyperparameters which are optimized and the optimization process are described.

### 4.3.2 Important Values

The following sections discuss important values for hyperparameter tuning.

#### 4.3.2.1 Learning Rate

Aditya Rakhecha describes the learning rate (LR) in "Towards Data Science" as follows: *"Learning rate (λ) is one such hyper-parameter that defines the adjustment in the weights of our network with respect to the loss gradient descent. It determines how fast or slow we will move towards the optimal weights."* [57]

The LR defines how fast the optimization algorithm moves towards optimal weights. A high LR can cause the model to converge too quickly and miss the optimal solution. On the other hand, a low LR can cause the opposite and leave a model stuck in a local minimum. [57]

In the original BERT paper, the authors selected the best LR among 5e-5, 4e-5, 3e-5, and 2e-5 for their fine-tuning of GLUE. [30]

#### 4.3.2.2 Weight Decay

Weight decay is a regularization term (L2) that penalizes large weights. It is also known as Ridge Regression [22]. In the Hugging Face library, the parameter `weight_decay` applies to all layers except bias and LayerNorm weights in the `AdamW` optimizer and defaults to `0` [58]. Often, a starting value of `0.01` is recommended [59]. The hyperparameter tuning search in this project is built around this value.

#### 4.3.2.3 Optimizer

As `AdamW` is the default optimizer in the `transformers` library [58], it is used in the fine-tuning process in this project. This optimizer is a combination of Adam and weight decay. It decouples weight decay from the optimization steps taken. This algorithm improves generalization and was proposed by Loshchilov and Hutter in 2017 [60].

#### 4.3.2.4 Batch Size

The batch size refers to the number of samples processed by the model at once during training. Goodfellow et al. suggest that this parameter should be more accurately called the *minibatch size* [22], as it represents a smaller subset of the entire training data, initially referred to as a batch. Despite this, the term batch size is commonly used interchangeably with minibatch size in both research literature and machine learning libraries. This thesis will also use the term *batch size* further on.

Table 12 shows the impact of batch size on training for large and small batch sizes.

| **Batch size** | **Large** | **Small** |
|---|---|---|
| **Memory Utilization** | Requires more memory | Requires less memory |
| **Training speed** | Faster | Slower |
| **Stability** | More stable, smoother updates | Less stable, adds noise |
| **Generalization** | Worse | Better |

Table 12: Impact of batch size on training [22], [61], [62]

Entry Point AI [63] is claiming that OpenAI dynamically configures the batch size to be `~0.2%` of the number of examples in the training set, but caps it at `256`. Also, OpenAI has found that larger batch sizes tend to work better on large datasets [64], [65]. Other sources recommend a batch size of `32` as a starting point. [66], [67]

Due to the large training dataset (chapter 6.1.2), the batch sizes of `16, 32, 64, 128` are included into the hyperparameter tuning search. The batch size of `256` is excluded because it exceeds available resources.

#### 4.3.2.5 Epochs

An epoch refers to a full pass through all available data, meaning the model has processed the entire dataset once. The number of epochs is a hyperparameter that dictates how many times the model will cycle through the dataset. To address overfitting, reducing the number of epochs may be beneficial, while increasing the epochs can help if the model is underfitting. [63]

Because of the big training dataset in this project (chapter 6.1.2), the need to revisit the same data frequently over many epochs is not necessary. This approach is also confirmed by Goodfellow et al.: *"With some datasets growing rapidly in size, faster than computing power, it is becoming more common for machine learning applications to use each training example only once or even to make an incomplete pass through the training set."* [22]

## 4.4 Statistical Evaluation

NER systems are typically evaluated using exact-match or relaxed-match metrics [68]. This thesis focuses on relaxed-match evaluation, counting partial matches as half correct.

### 4.4.1 Exact-Match Evaluation

Exact-match evaluation compares the system's output to the ground truth annotations, considering an entity as correct only if it is an exact match. [11] The following terms are used:

**False Positive (FP)** Returned by NER but not in ground truth
**False Negative (FN)** In ground truth but not returned by NER
**True Positive (TP)** Returned by NER and in ground truth

These terms are used to calculate precision, recall, and F1 score.

**Precision** How many of the recognized entities are correct
**Recall** How many of the actual entities were found by the system
**F1 score** Harmonic mean of precision and recall

The formulas for these metrics are as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{F1} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

### 4.4.2 Partial-Match Evaluation

A common tool for evaluating NER systems is the `nervaluate` Python package [69]. It provides an easy way to evaluate models in different ways. The package calculates precision, recall, and F1 score for each entity type and overall. Additionally, it allows to specify whether partial matches should be counted as correct or not.

When it comes to citation recognition, partial matches are better than no match at all. Thus, the `partial` evaluation method is used. This method counts partial matches as half correct. In this way, the model is not punished twice for a false positive and a false negative. The calculations for precision and recall are adjusted as follows:

$$\text{Precision} = \frac{(\text{COR + 0.5 x PAR})}{\text{ACT}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{(\text{COR + 0.5 x PAR})}{\text{POS}} = \frac{\text{COR}}{\text{ACT}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

### 4.4.3 Bucket-Wise Evaluation

To further understand the model's performance, bucket-wise evaluation can be applied. This helps to identify with which buckets the model struggles. It allows optimizing the model for specific subsets of the data. [70]

Possible buckets for the evaluation are:

- **Language**: The test data could be split into buckets based on the attribute language. This may help to identify potential biases in the model and to improve its performance on specific languages.
- **Citation Type**: Another split could be into buckets based on the citation type, such as case law, literature, or law. This could help to identify for which citation types the model performs well and for which it does not.

## 4.5 Processing Pipeline

Since the task of recognizing citations and their parts can be divided into two subtasks, different processing pipelines are possible. These range from a single model handling everything to multiple models working in sequence or parallel. As the choice of architecture depends, among other factors, on the availability of training data, the current data situation is outlined first. In the following sections, potential architectures for the processing pipeline are discussed.

### 4.5.1 Data Availability

As analyzed in chapter 3.3, there is more than enough labeled data available for recognizing citations. This makes it possible to fine-tune a model specifically for this task. However, the availability of detailed annotations for individual parts of a citation, such as articles or page numbers, is highly limited, as described in chapter 3.3.3. In order to fine-tune a model, training data would first need to be generated for the parts. Hence, few-shot prompting with a pre-trained language model (chapter 4.2.2.2) becomes a possible alternative.

### 4.5.2 Single Model

One approach is to treat the task of recognizing citations and their parts as a single problem. The document is passed to one model, which recognizes both citations and their parts in the same step. This approach is illustrated in Figure 15.


Figure 15: Single model pipeline

Recognizing both citations and their detailed parts in one step requires the model to handle a complex task. This might be too demanding for smaller models, leading to worse performance. Larger models like Llama may be needed to handle this complexity. However, larger models require more resources such as memory, storage space, and computing power. Inference time also gets slower. In addition, it is not possible to post-process the model output for each subtask separately. This means that errors or inaccuracies when recognizing citations might propagate to the task of recognizing parts. The output is not easy to debug and less explainable.

An advantage of this approach is that maintaining only one model simplifies deployment.

### 4.5.3 Model Chaining

The model chaining approach splits the task of citation recognition into two sequential steps. The first model identifies the citation entities in the text. It then passes the text fragments to a second model that extracts the individual components of each citation. This process is known as model chaining, as further explained in chapter 4.2.5. The architecture is visualized in Figure 16.


Figure 16: Model chaining pipeline

This approach offers several advantages over using a single model. By dividing the task, each model can be optimized for its specific subtask, potentially leading to better overall performance. Being fine-tuned, the first model can be more accurate in recognizing citations. The second model benefits from receiving only the relevant portions of text as input, simplifying its task.

Additionally, post-processing becomes feasible after the first task, allowing for corrections or refinements in citation recognition before extracting detailed components. This staged process also means that the context size required for the second model is reduced, which could improve its efficiency. The output of each model can be analyzed separately, making debugging and error analysis easier. It has a high explainability.

However, this approach also comes with some disadvantages. Managing two models requires additional resources, such as memory and storage space, and increases power consumption. Deployment becomes more complex, as updates or changes need to be coordinated across multiple models. Inference time could also be longer. However, this depends on the models used and the communication overhead between the stages.

### 4.5.4 Model Chaining with Multiple Classes

Another approach is to divide the task again into two steps, with the first model performing a more detailed classification. Instead of just recognizing `CITATION` entities, it identifies different types like `LAW`, `CASELAW`, and `LITERATURE`. This architecture is shown in Figure 17.

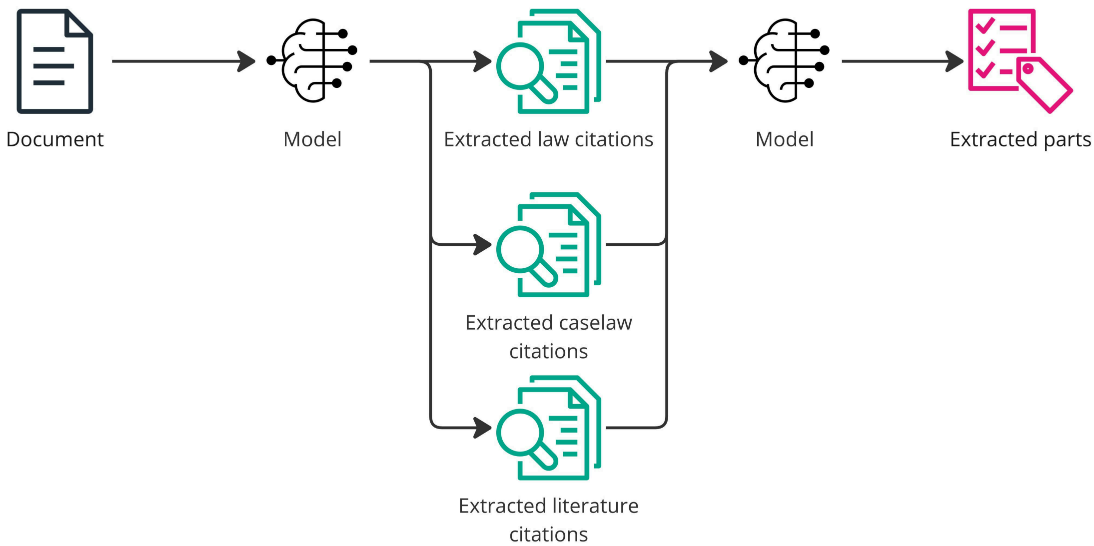



Figure 17: Model chaining with distinct classes pipeline

Additionally to the advantages of the chaining approach in chapter 4.5.3, this variant offers more flexibility and adaptability. The structure of citations can vary depending on the type of entity. Defining multiple classes can help the model better understand the features of each entity. It also provides more information about the citation, which can be helpful when linking it to the correct source later.

Using multiple citation classes can also enhance the second model's performance. With few-shot prompting, the prompt to extract the parts can be adjusted based on the type of citation. Different examples can be provided for each class. Since only the relevant domain-specific examples are provided, the prompt gets shorter, leading to faster inference times.

However, classifying entities into finer-grained types requires a more capable first model and potentially larger amounts of annotated training data. This might not be available for all classes. The added specialization also means that the software becomes more complex, as it needs to dynamically adjust the prompt based on the classification result.

### 4.5.5 Model Chaining with Multiple Classes and Specialized Models

Another variant of the model chaining approach is to use specialized models for parts extraction for each class of entities. The first model classifies entities into different types. Each type is then processed by a dedicated specialized model. The proposed architecture is visualized in Figure 18.

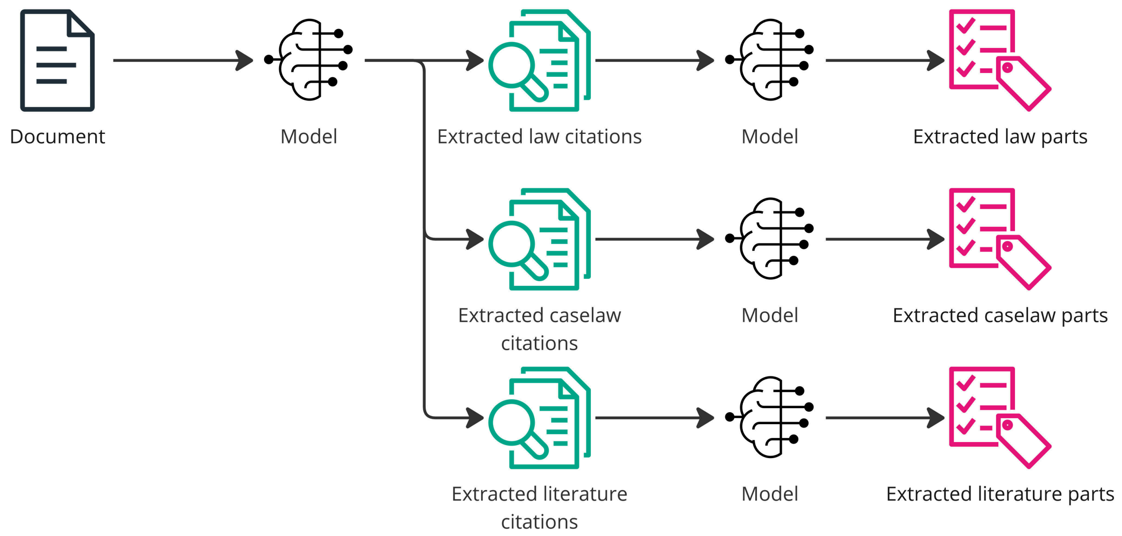



Figure 18: Model chaining with multiple classes and specialized models pipeline

However, this approach is not practical for this project. The types of parts are not exclusive to specific entity classes. For instance, the part type `VOLUME` can appear in both `LITERATURE` and `CASELAW` citations. In such cases, a single model for all classes is more effective. Moreover, if few-shot prompting with a pre-trained model is used, having specialized models becomes redundant.

### 4.5.6 Ensemble of Models

An ensemble of models is another possibility for citation recognition. It can be used in combination with any of the previously described approaches. Multiple models are trained independently. They either each focus on a specific aspect of the task, or use different architectures or hyperparameters.

The outputs of the models are combined. Which prediction to use can be determined by a voting mechanism, by averaging the predictions, or with other aggregation methods. The approach is illustrated in Figure 19.

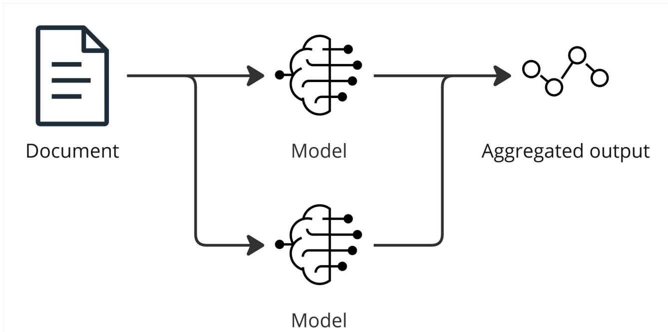



Figure 19: Ensemble of models pipeline

# 5 Design

This chapter presents the solution. It covers the processing pipeline, architecture, data schema, fine-tuning process, and technologies used.

## 5.1 Choice of Processing Pipeline

As shown in chapter 4.1, various approaches to NER exist. The most effective methods for citation recognition leverage deep learning models like BERT or LLMs. They have demonstrated high performance across numerous NLP tasks, including NER.

Due to the NFRs specified in chapter 2.2, it is not possible to use GPT-4o from OpenAI. The model is hosted externally, which poses a security risk and may lead to data privacy issues. The API is not cost-effective, as every request to the API costs money. Lastly, due to the external hosting, there is a risk of vendor lock-in.

Training a model from scratch would require a large amount of data, computational power, and time, which is not available in this project, nor is it necessary. The pre-trained models can be fine-tuned on a specific dataset to achieve good results.

One possibility is to solve the task of recognizing citations and parts in one step with a single model. For this, few-shot prompting on a Llama model is evaluated. However, it quickly becomes apparent that this approach is not feasible. The model is not able to process the required 3000 documents per night. It would take approximately 32 hours for that amount. The performance is also not satisfactory. The experiment is described in more detail in appendix section F.2.6.

This leads to the decision to split the task into two subtasks: recognizing citations and in a second step, recognizing the specific parts. For each subtask, a different model is used. The output from the first model is passed to the second model (chapter 4.5.4).

To recognize citations, a fine-tuned model (chapter 4.2.3) is used. Multiple models are evaluated. Another evaluation is done about the possibility of using multiple models in parallel and aggregating their results (chapter 4.5.6).

Due to the disparity in data availability described in chapter 4.5.1, it is not possible to directly fine-tune a model on the task of recognizing parts of a citation. Either few-shot prompting (chapter 4.2.2.2) or knowledge distillation (chapter 4.2.4) have to be used. Both of these approaches are evaluated.

## 5.2 Solution Architecture

The architecture of the solution is based on the requirements described in chapter 2. The following chapters describe the architecture based on the C4 model. The C4 model is a hierarchical model for visualizing the architecture of software systems. It consists of four levels: context, container, component and code. The code level is ommitted in this thesis, as such a detailed view is not necessary. [71]

### 5.2.1 C4 - Context Layer

The context diagram can be seen in Figure 20. The *Citation Recognition* solution is part of the Swisslex ecosystem and interacts with the *Job Scheduler* and the *Database* of Swisslex.

**Swisslex Developer** Uses the solution, develops and runs the citation recognition.

**Job Scheduler** External tool which runs different jobs periodically. One of these jobs will be the `CitationRecognitionJob`, which is a part of the *Citation Recognition* software system.

**Database** The *Database* is used by the solution to read documents and store the recognized citations and its parts. The existing database schema is described in chapter 3.1.4.3. The extended tables used for citation recognition are described in chapter 5.4.

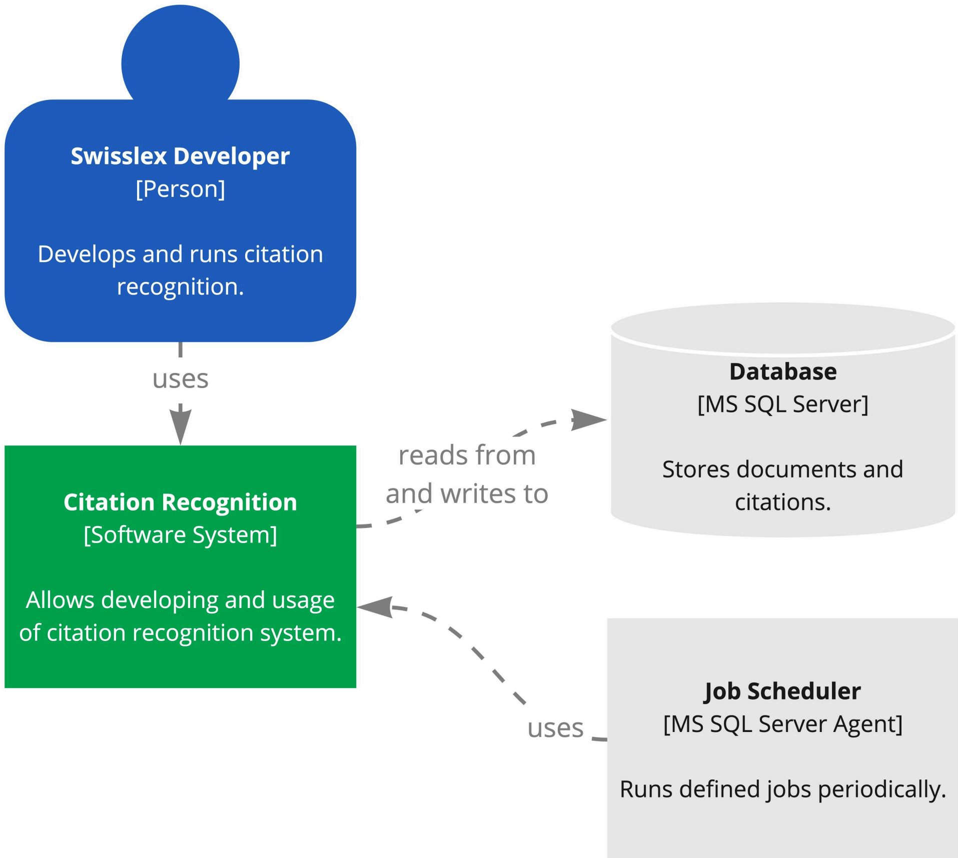



Figure 20: C4 context diagram

### 5.2.2 C4 - Container Layer

The container diagram can be seen in Figure 21. The software system *Citation Recognition* contains the following containers:

**Citation Checker Web** The web application can be used to test the citation recognition (chapter 6.3).

**Citation Processor** This project contains the `CitationRecognitionJob` which can be called by the *Job Scheduler* from Swisslex. It can also be used as a console application to processes multiple documents (chapter 6.4).

**Parts Training Data Generator** This console application is used to generate training data for the recognition of the parts of a citation with an LLM (chapter 6.2.3.1).

**Citation Inference API** This Python application uses FastAPI to provide an API for the fine-tuned models (chapter 6.5).

**Data Extractor** This project is used to extract data from the *Database* and generate the datasets needed to fine-tune models (chapter 6.1.2.3).

**Python Project** This project contains the code to fine-tune the models and evaluate them.

**vLLM Server** With vLLM, an inference server for LLM base models can be started. This is used for few-shot prompting and knowledge distillation (chapter 6.2.2 and chapter 6.2.3.1).

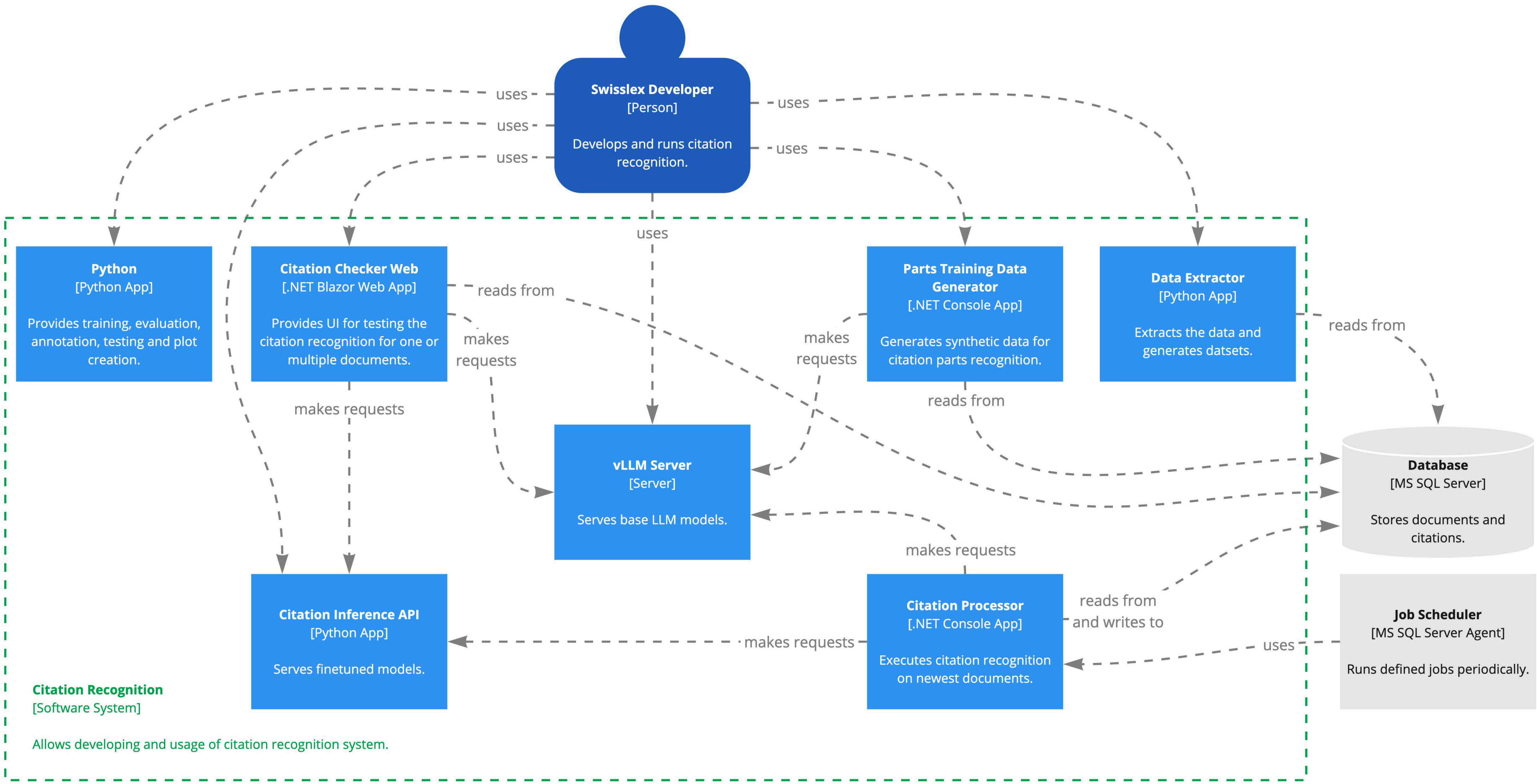



Figure 21: C4 container diagram

### 5.2.3 C4 - Component Layer

The component diagrams for the different containers can be seen in the following sections.

#### 5.2.3.1 Citation Checker Web

The component diagram for the website *Citation Checker Web* can be seen in Figure 22. It contains the following components:

**CitationCheckerWeb** This project is a .NET Blazor Web App and functions as the frontend. It is used to test the citation recognition.

**CitationCore** This project is a .NET Standard library that encapsulates the core logic shared across all .NET projects. It is used to recognize citations and their parts by calling the *vLLM Server* or the *Citation Inference API*. Additionally, it includes the functionality to retrieve documents from the database.

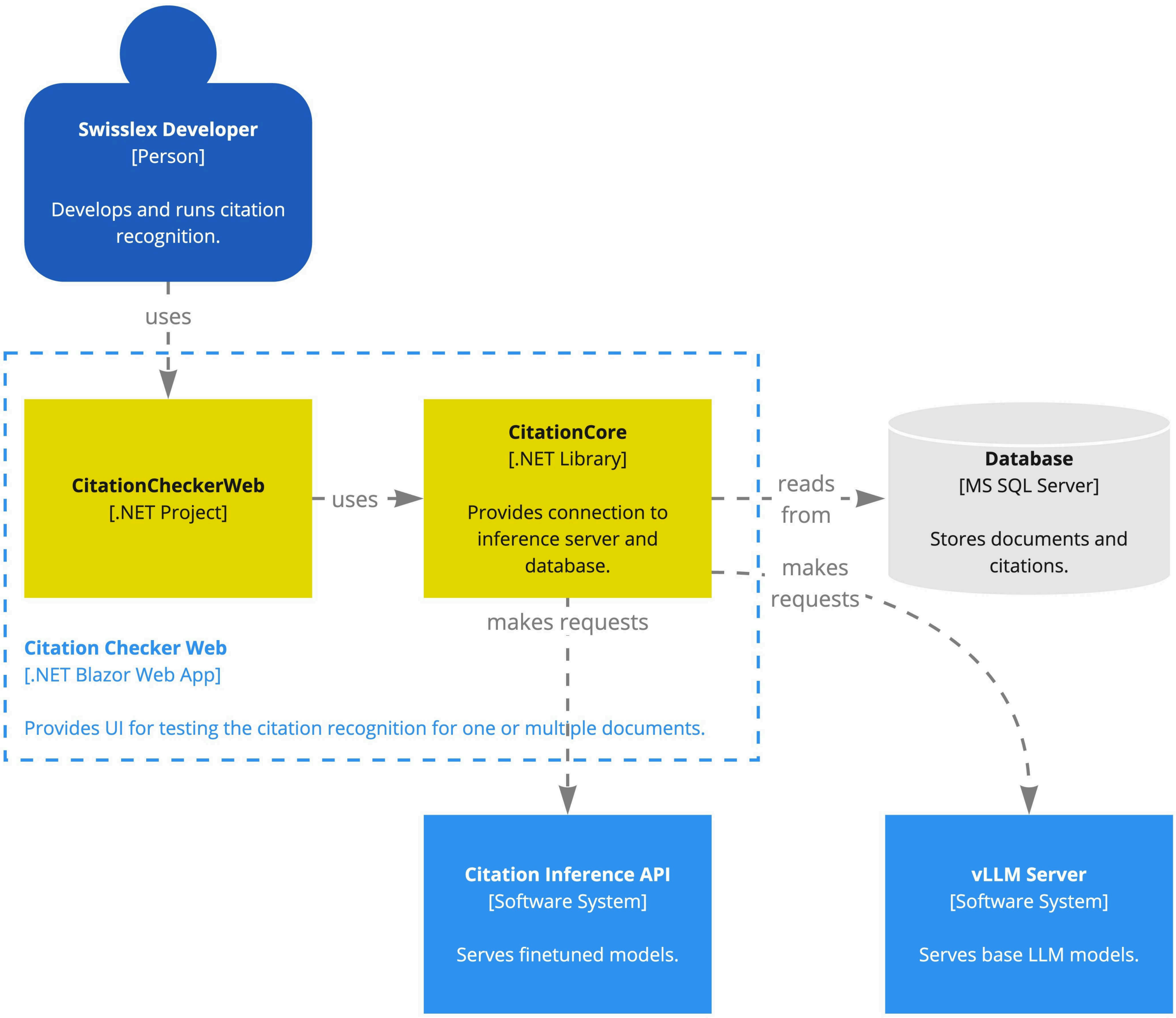



Figure 22: C4 component diagram - Citation Checker Web

#### 5.2.3.2 Citation Processor

The component diagram for the application *Citation Processor* can be seen in Figure 23. It contains the following components:

**CitationProcessor** This project is a .NET Console Application which can be used to run the `CitationRecognitionJob`.

**CitationCore** This project is a .NET Standard library that encapsulates the core logic shared across all .NET projects. It is used to recognize citations and their parts by calling the *vLLM Server* or the *Citation Inference API*. Additionally, it includes the functionality to retrieve documents from the database. It is used to by the *CitationProcessor*. After recognizing citations and parts, they are stored in the *Database*.

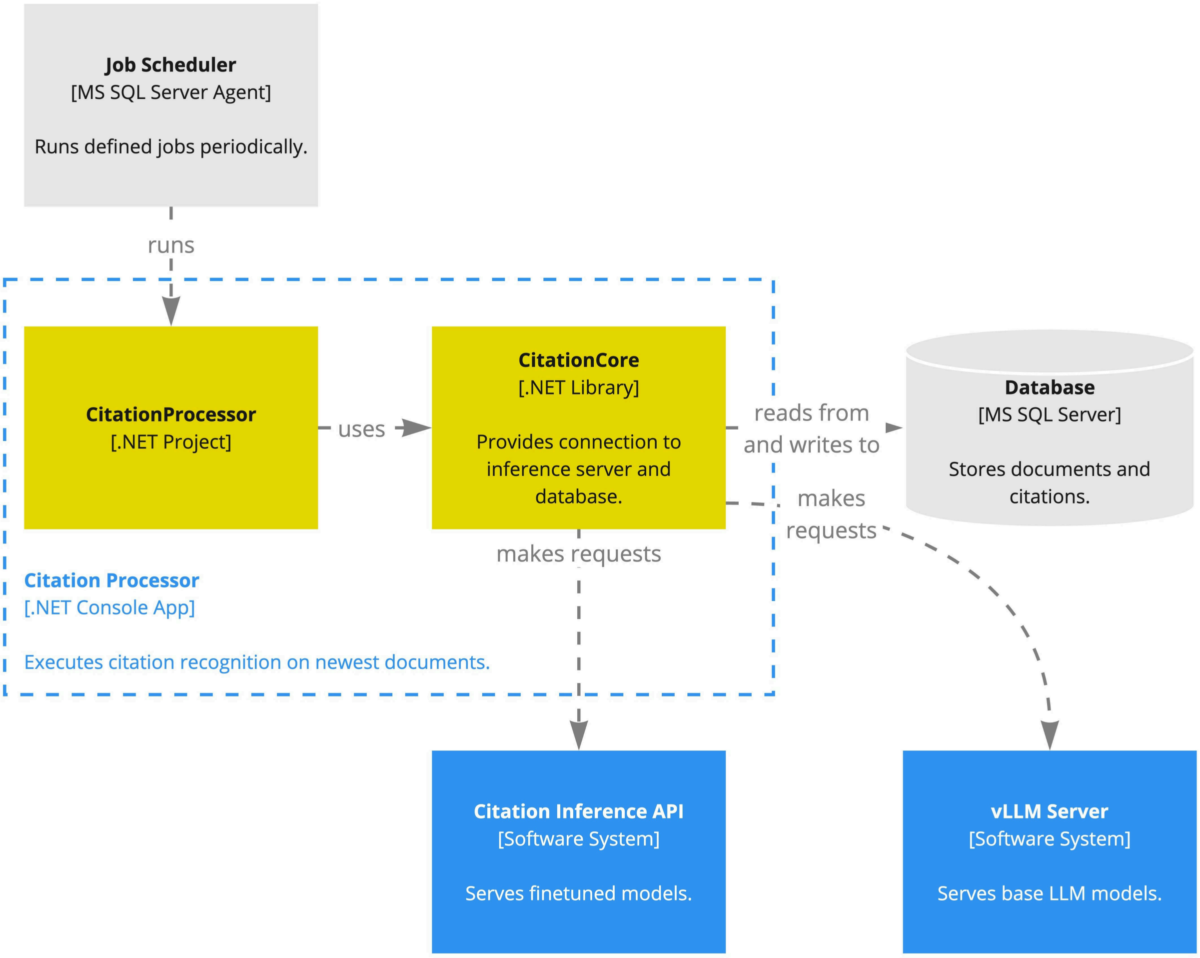



Figure 23: C4 component diagram - Citation Processor

#### 5.2.3.3 Parts Training Data Generator

The component diagram for the application *Parts Training Data Generator* can be seen in Figure 24. It contains the following components:

**PartsTrainingDataGenerator** This project is a .NET Console Application which can be used to generate synthetic training data for citation parts recognition.

**CitationCore** This project is a .NET Standard library that encapsulates the core logic shared across all .NET projects. It is used to by the *PartsTrainingDataGenerator* to recognize parts from citations. The citations are read from the *Database*. The generation is done by using few-shot prompting on an LLM served by the *vLLM Server*.

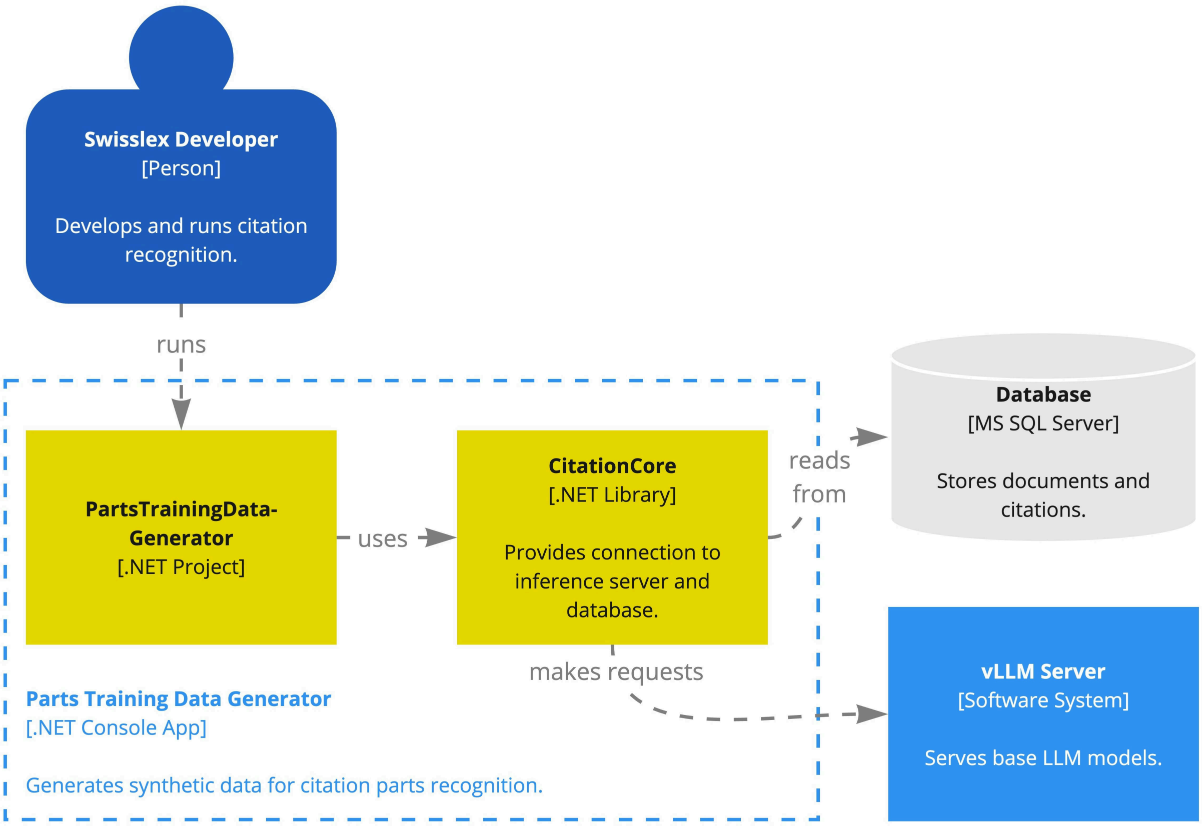



Figure 24: C4 component diagram - Parts Training Data Generator

#### 5.2.3.4 Citation Inference API

The component diagram for the application *Citation Inference API* can be seen in Figure 25. It contains the following components:

**app.py** is the script to start the server.

**controllers** contains the logic to receive the requests and to call the *services*.

**utils** contains helper functions like sliding window logic and post processing.

**services** contains the necessary logic to load and use the models

**models** contains classes used by the *services* and *controller* component.

**configs** contains the list of models which can be served.

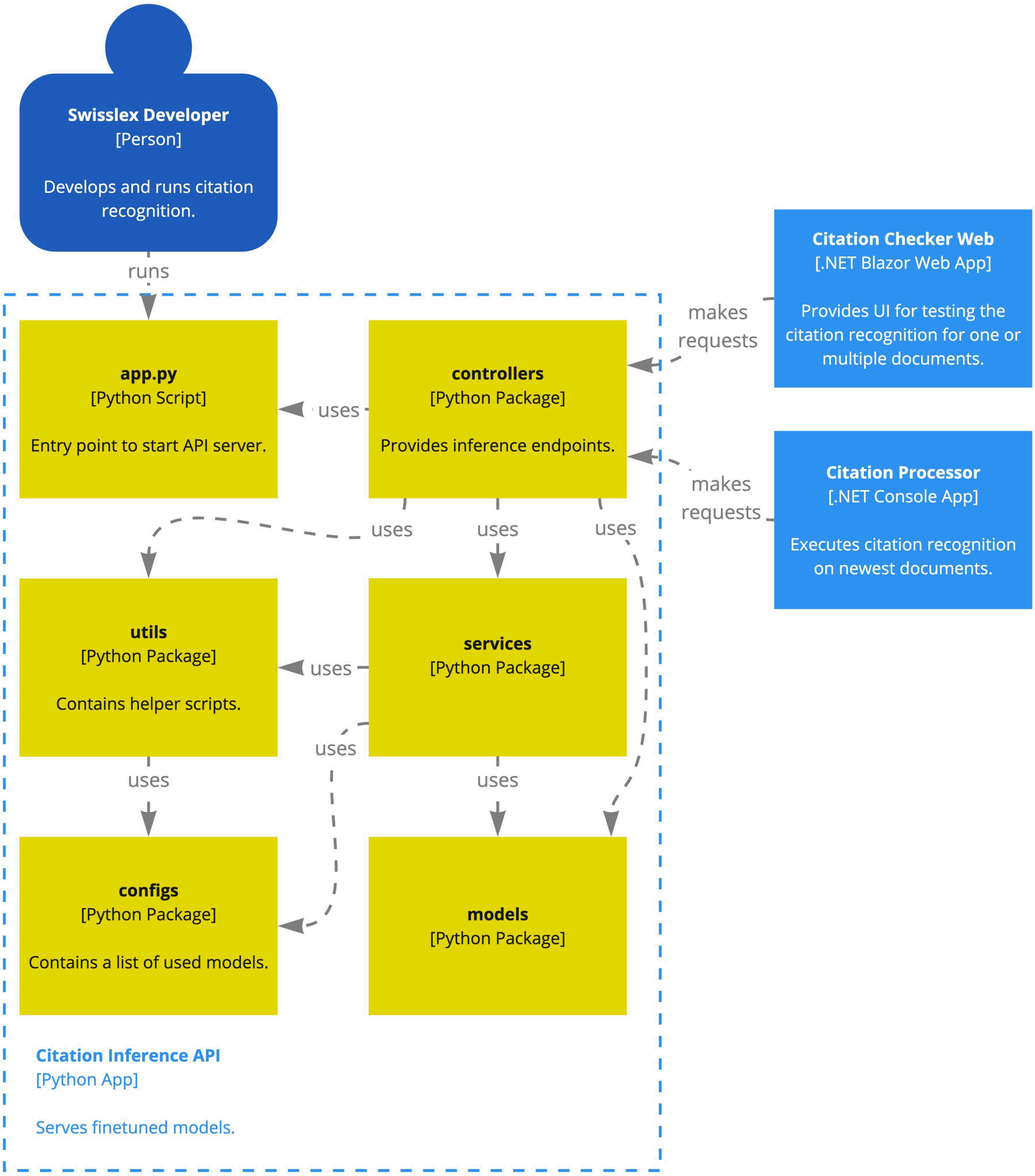



Figure 25: C4 component diagram - Citation Inference API

#### 5.2.3.5 vLLM Server

The component diagram for the application *vLLM Server* can be seen in Figure 26. It contains the following components:

**vLLM Server** The server can be started with a bash command. The running instance is used by the *Parts Training Data Generator* to generate synthetic training data for citation parts recognition. The *Citation Checker Web* also uses the server when a user chooses few-shot prompting as recognition mode. The *Citation Processor* can also use it if the developer decides to use few-shot prompting to recognize parts.

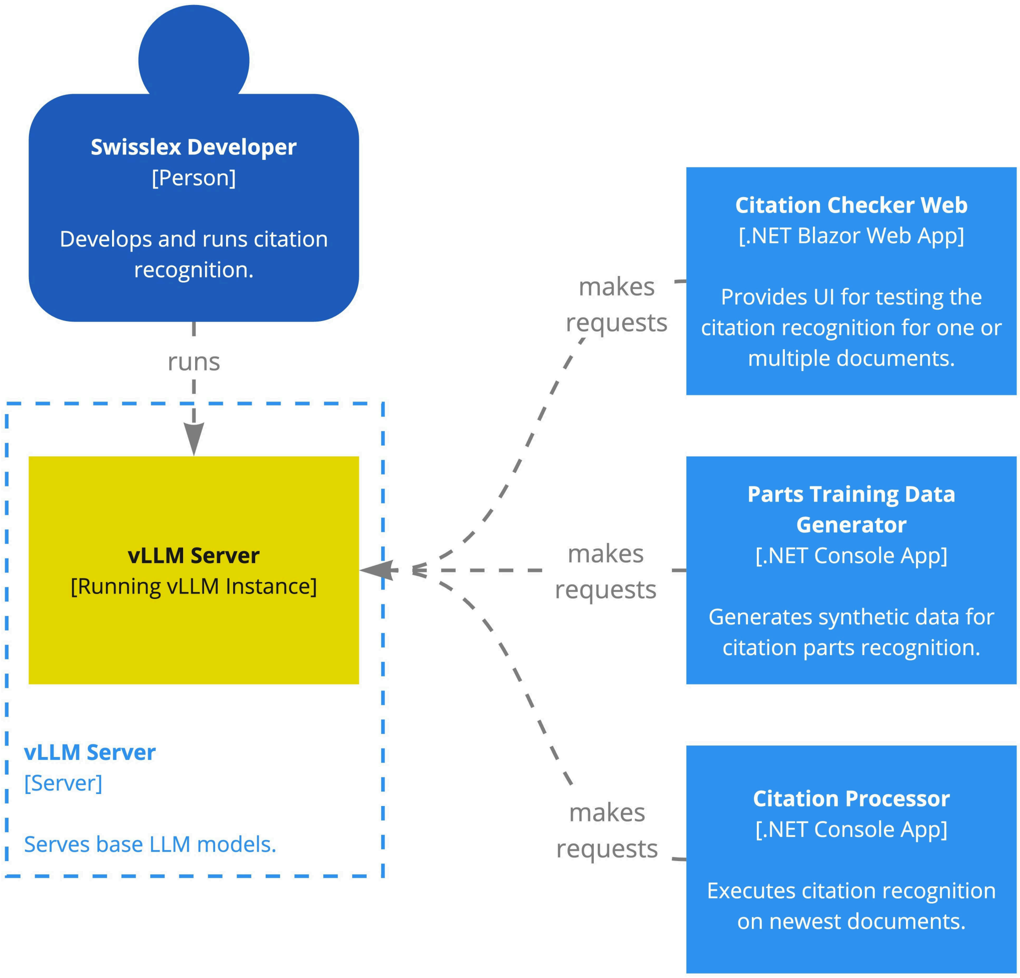



Figure 26: C4 component diagram - vLLM Server

#### 5.2.3.6 Data Extractor

The component diagram for the application *Data Extractor* can be seen in Figure 27. It contains the following components:

**cli.py** The Python script to run the data extraction pipeline.

**extractor.py** This script contains the logic of the data extraction pipeline. This includes SQL queries and the merging of the extracted data.

**helper** This package provides helper functions for data extraction, such as the database connection.

**resources** Contains the IDs used for the golden test dataset and the special cases dataset.

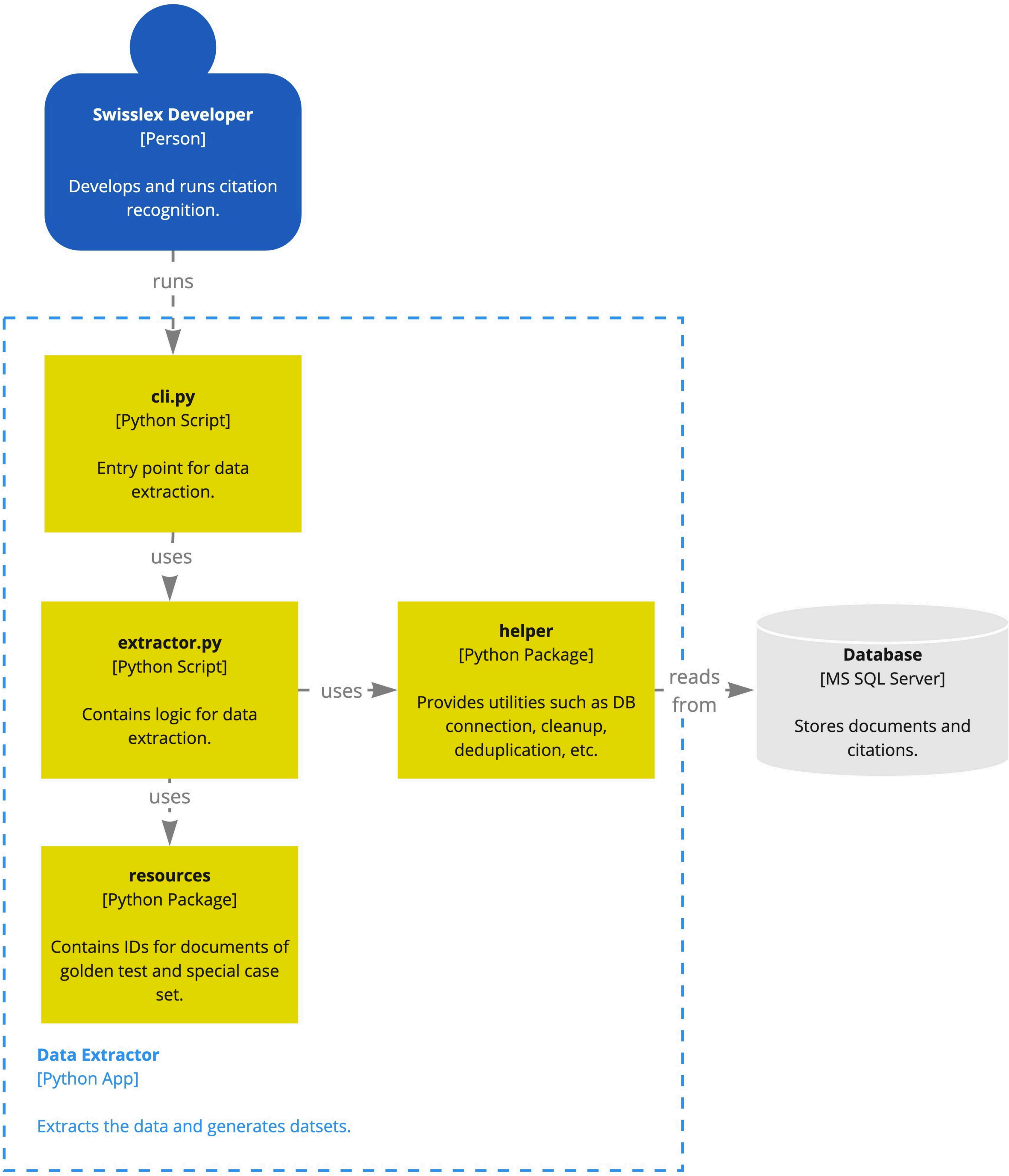



Figure 27: C4 component diagram - Data Extractor

### 5.2.3.7 Python Project

The component diagram for the application *Python Project* can be seen in Figure 28. It uses different packages to test functionality, annotate the golden and special case datasets, create plots, and train and evaluate models.

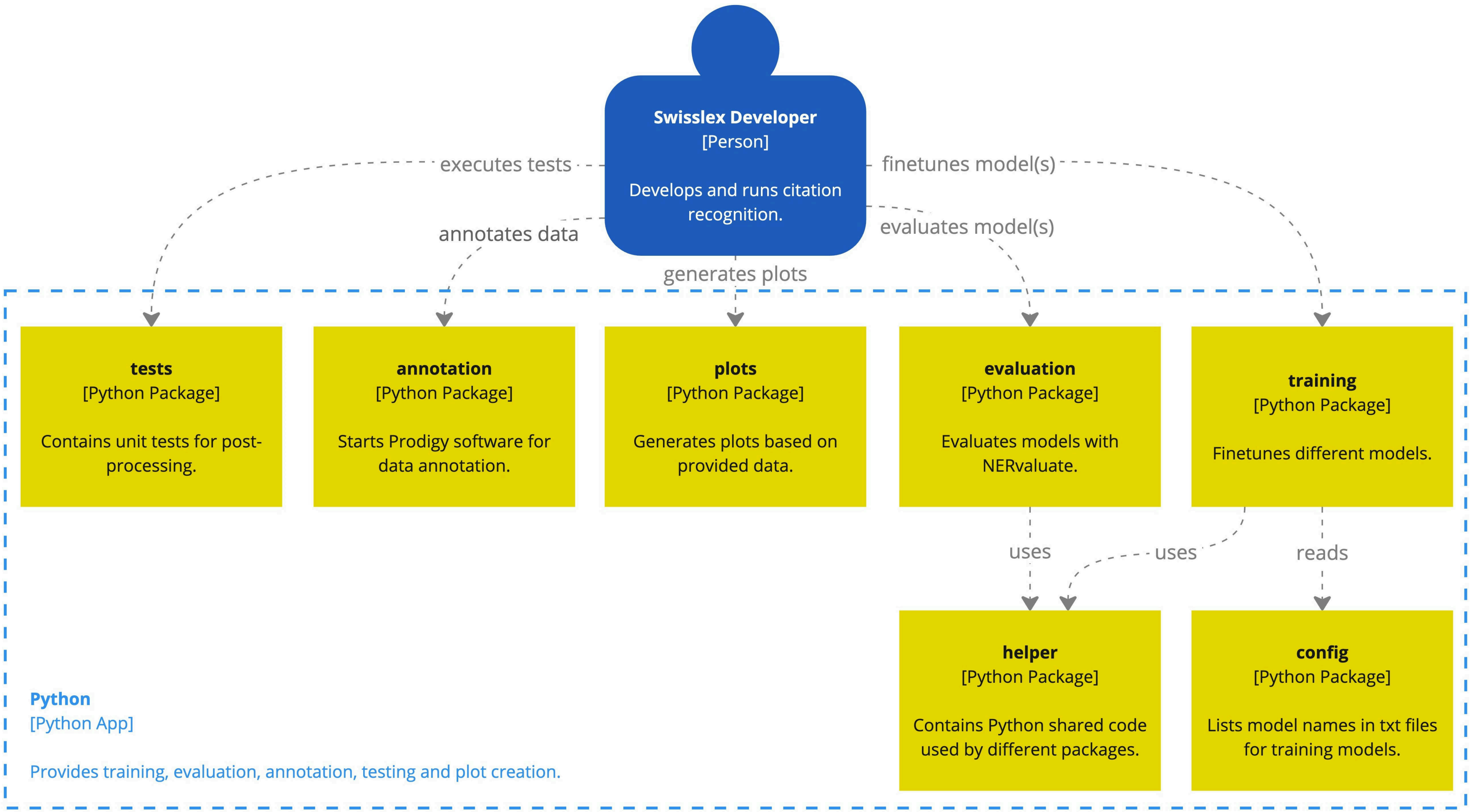



Figure 28: C4 component diagram - Python Project

## 5.3 Architectural Decisions

This chapter documents the architectural decisions that led to the current solution architecture in chapter 5.2. To document them in a lean and structured way, the Y-Statement template by O. Zimmermann is used [72].

### ADR-001: Categorization of citations in LAW, CASELAW and LITERATURE

| | |
|---|---|
| *In the context of* | categorizing legal citations into distinct types |
| *facing* | the challenge of handling different structures of citations |
| *we decided* | to divide citations into the categories `LAW`, `CASELAW`, and `LITERATURE` |
| *and neglected* | a more granular or unified categorization approach |
| *to achieve* | a clearer structure and improved performance in downstream tasks |
| *accepting* | the risk of sometimes not 100% clear categorization in one of the three categories |

### ADR-002: Create fine-grained parts of a citation

| | |
|---|---|
| *In the context of* | improving the recognition of parts of a citation |
| *facing* | the challenge of diverse structures of citations |
| *we decided* | to create fine-grained parts of a citation like `ARTICLE`, `PARAGRAPH`, and `PAGE` |
| *and neglected* | a simpler, less granular approach to parts labeling |
| *to achieve* | better adaptability to nuanced citation structures and increased accuracy in downstream tasks |
| *accepting* | the additional effort required to annotate a more detailed dataset for training |

### ADR-003: Starting possibilities of citation recognition in UI

| | |
|---|---|
| *In the context of* | starting the citation recognition process through the UI |
| *facing* | the problem of not knowing which of both starting points to implement |
| *we decided* | to give the user a document-upload and dossier number field to start the citation recognition process |
| *and neglected* | presenting only one or the other |
| *to achieve* | good and flexible usability and having the possiblity to upload any text (no dependencies to the SLX database) |
| *accepting* | a more complex UI and more complex backend logic |

### ADR-004: Use of BERT-based approach to recognize parts of a citation

| | |
|---|---|
| *In the context of* | recognizing parts of a citation |
| *facing* | the need to process 3,000 documents within 6 hours |
| *we decided* | to use a BERT-based approach during inference |
| *and neglected* | few-shot prompting based on Llama |
| *to achieve* | a high throughput while maintaining a comparable accuracy |
| *accepting* | having to create a new parts dataset from scratch for training |

## 5.4 Database Schema

To store the found citations, a database schema is created. It extends the database from Swisslex that is described in chapter 3.1.4.3.

The new schema is shown in Figure 29. It consists of the following tables:

**`Citation`** Stores all citations found by the model, including start and end positions in the document, the found text (passage), the type (`CASELAW`, `LAW` or `LITERATURE`) and the confidence of the model. It is linked to the `AssetBase` table and points to the document in which the citation was found.

**`PartType`** Stores the different types of citation parts that can be found, such as `COURT`, `LAW`, `ARTICLE` or `CONSIDERATION`. The values are predefined. They are listed and described in detail in chapter 6.2.1.

**`CitationPart`** Stores the values of the different parts of a citation and links them to their respective `Citation` and `PartType` entries. Additionally, the start and end positions of the part in the citation are stored, as well as the confidence of the model for this part.

The `AssetBase` table already exists in the Swisslex database and is used as an entry point for all documents in the system. It contains metadata about the document, such as the title, the document type, the publication date, and the document ID.

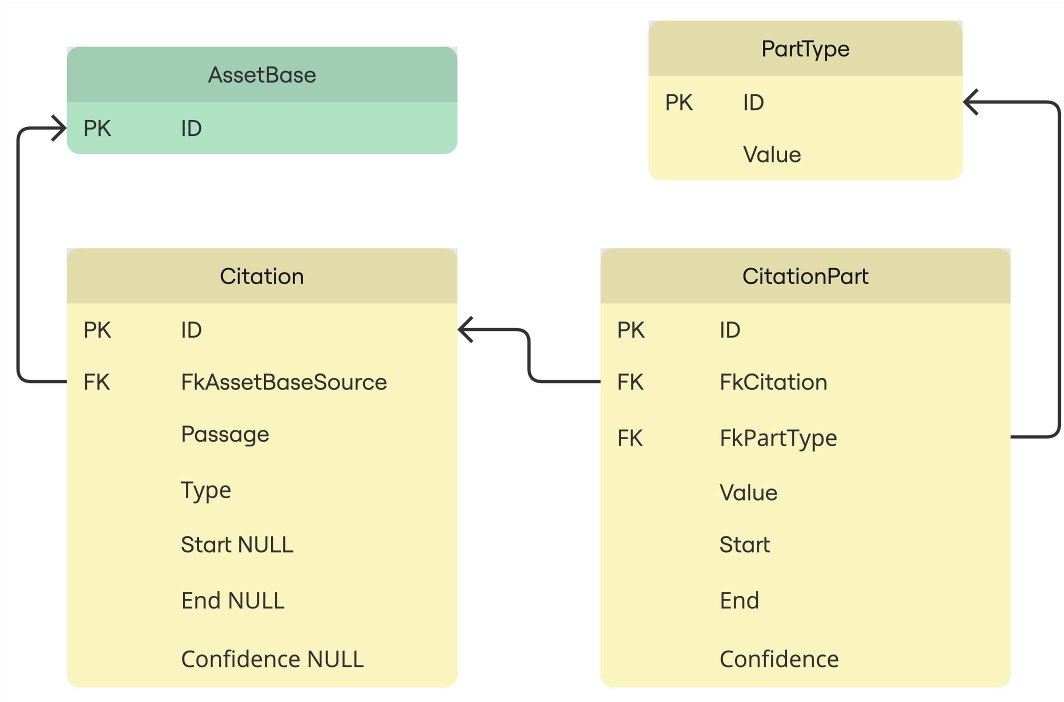



Figure 29: Database schema to store the citations

## 5.5 Used Technologies

Figure 30 shows the technology stack used in this project.

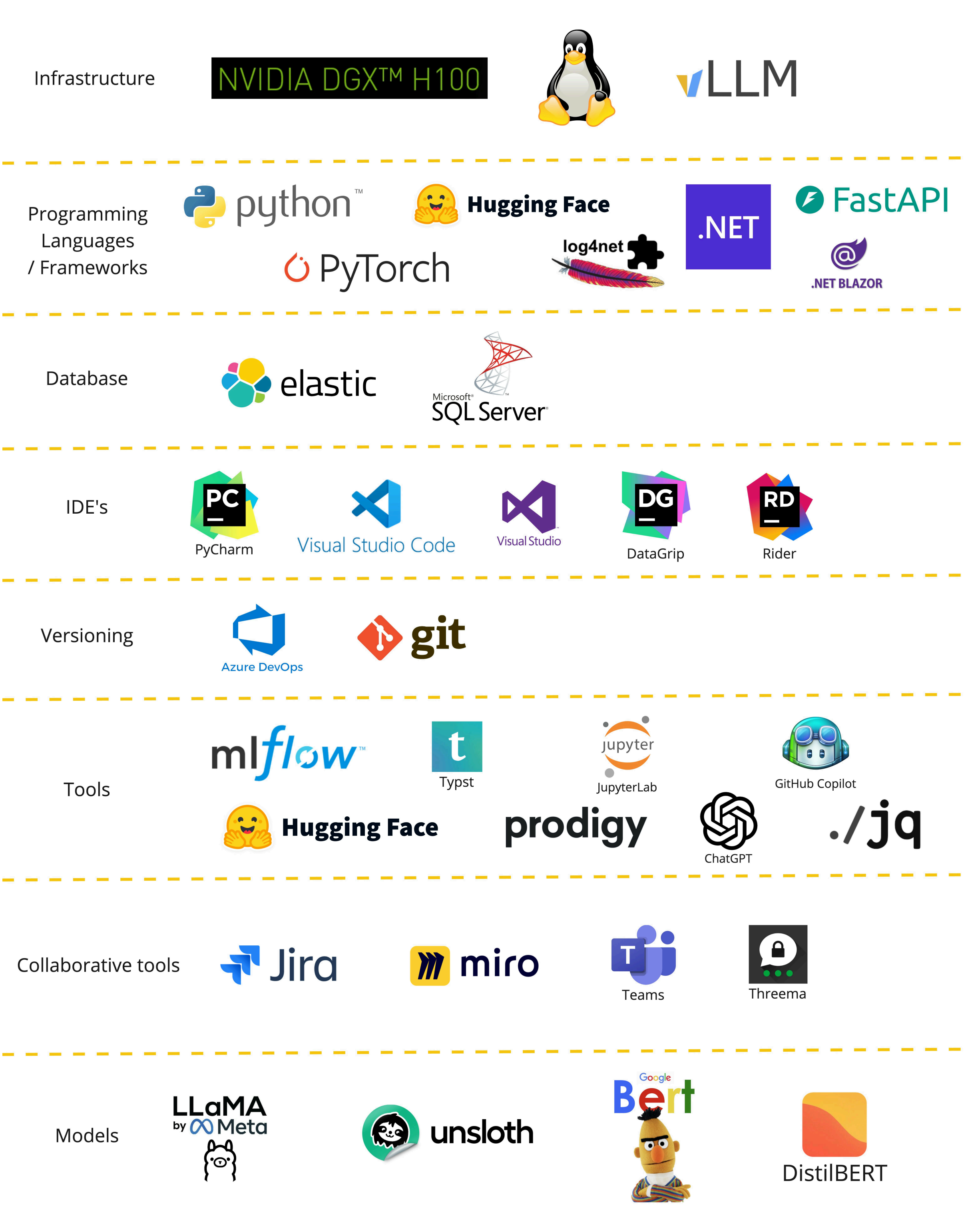



Figure 30: Technology stack

### 5.5.1 Infrastructure

The components of the infrastructure are described in the next sections.

**NVIDIA DGX H100** The NVIDIA DGX H100 [73] is a powerful system with 8 H100 GPUs, designed to accelerate enterprise AI workloads. During this project, access is provided to one full H100 GPU and one H100 GPU shared across three JupyterLab instances, running on a Linux-based operating system.

**vLLM** To serve the Llama model for inference, vLLM [74] is used. vLLM is an optimized inference library for large language models.

### 5.5.2 Programming Languages / Frameworks

Different programming languages and frameworks are used in this project, as described in the following sections.

**Python** [75] is a versatile programming language that is widely used in the field of data science and machine learning. It is used for the implementation of the models, the data processing, and the training of the models.

**PyTorch** [76] is an open-source machine learning library based on the Torch library. It is used for applications such as natural language processing. In this project, PyTorch is used to implement the models and to train them.

**Hugging Face Libraries** Hugging Face [77] provides various useful libraries such as Transformers, Tokenizers, and Datasets. The Transformers library is used to load and fine-tune the pre-trained models, the Tokenizers library to tokenize the input data, and the Datasets library to load the data.

**FastAPI** [78] is a modern, fast web framework for building APIs with Python. FastAPI is used to serve the BERT based models for inference.

**Microsoft .NET** [79] is a free, open-source, cross-platform framework for building modern, cloud-based, internet-connected applications. .NET is used to implement the core logic of the citation processor job that is executed on a daily basis.

**Blazor** [80] is a web framework to create user interfaces for web apps using C#. Blazor is used to provide a user interface for developers to check the performance of the models.

**Log4Net** [81] is a flexible and widely used logging framework for .NET applications.

### 5.5.3 Database

The database systems used in this project are described in the following sections.

**Elasticsearch** [82] is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. Since the Swisslex documents are also available in Elasticsearch, it is used to gain a quick insight into the data.

**Microsoft SQL Server** Swisslex uses Microsoft SQL Server [83] as their database system. Microsoft SQL Server is a robust relational database management system for storing, querying, and managing data at scale. The database is used to retrieve data for our training and testing of the models and to store the final results of the parts of a citation.

### 5.5.4 IDEs

Various IDEs are used, including PyCharm [84] for Python development, Visual Studio [85] and JetBrains Rider [86] for .NET development, DataGrip [87] to interact with the database and Visual Studio Code [88] to write the documentation.

### 5.5.5 Versioning

Azure DevOps [89] is used to manage the versioning of the code and the documentation. Azure DevOps is a set of services for teams to share code, track work, and ship software.

### 5.5.6 Tools

The various other tools used in this project are described in the following sections.

**JupyterLab** [90] is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is used to be able to run the code on the provided GPUs.

**MLflow** To track the experiments and the model performance, MLflow [91] is set up. MLflow is an open-source platform for managing the end-to-end machine learning lifecycle.

**Prodigy** To label the golden and special case dataset, Prodigy [92] is used. Prodigy is an annotation tool for creating training data for machine learning models.

**Typst** [93] is a modern markup-based typesetting system for creating high-quality documents with ease and flexibility. In this project, Typst is used to write the documentation.

**ChatGPT** ChatGPT [94] is utilized for coding assistance, text rephrasing, and formatting adjustments.

**GitHub Copilot** GitHub Copilot [95] is used for coding assistance.

**jq** [96] is used for processing and inspecting JSON data.

### 5.5.7 Collaborative Tools

Microsoft Teams [97] and Threema [98] are used for communication and collaboration, Miro [99] for brainstorming and planning, and Jira [100] for task management and time tracking.

### 5.5.8 Models

To recognize the citations and the parts of a citation, the following models are evaluated.

**BERT** The performance of the citation recognition is evaluated on a variety of BERT flavours, including original Google BERT [30], DistilBERT [101], DeBERTaV3 [102] and XLM-RoBERTa [103].

**GliNER** The performance of the citation recognition is also evaluated on a fine-tuned GliNER [38].

**Llama** The Unsloth Llama 3.2 1B Instruct [104], Llama 3.2 3B Instruct [105] and Llama 3.1 8B Instruct [106] are fine-tuned for citation recognition, in order to compare them to the performance of the fine-tuned BERT models. Additionally, the Unsloth Llama 3.1 8B Instruct and Llama 3.3 70B Instruct [107] model are used for the few-shot prompting approach to recognize parts and the generation of the parts training data.

# 6 Implementation

In this chapter, the implementation of the solution is described. Figure 31 shows the a simplified structure of the recognition process. First, a sample from the chunked document is processed by a fine-tuned Google BERT model, which identifies citations categorized as `CASELAW`, `LAW`, or `LITERATURE`. The recognized citations are then sent to a fine-tuned DistilBERT model, which recognizes the parts of the found citations.

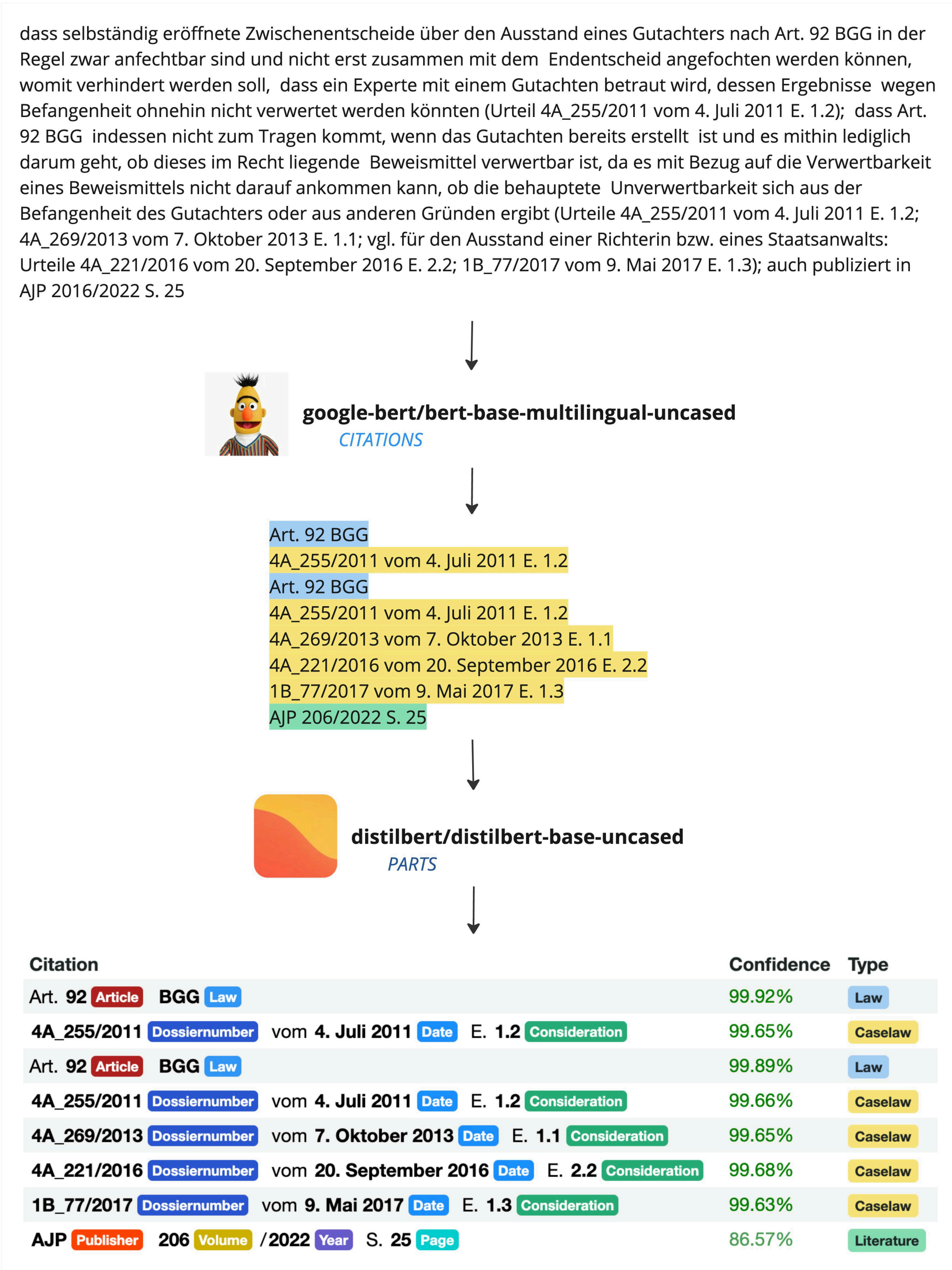



Figure 31: Overview diagram

## 6.1 Recognizing Citations

This chapter describes the fine-tuning and evaluation of different models for recognizing citations. After a hyperparameter optimization, various base models are fine-tuned on a small dataset of 100′000 samples. The most promising models are further evaluated after training on more samples. In the end, the performance per label is evaluated as well as the performance on special cases. Lastly, the speed of the models is measured.

### 6.1.1 Label Selection

Given the variety of citation styles in legal texts, it is better to adopt specific labels than to use a single, generic `CITATION` label. The advantages of multiple classes are listed in chapter 4.5.4. The labels should represent the distinct types of citations encountered in the data. In the context of legal documents, the most common types of citations are references to laws, judicial decisions, and literature citations. Therefore, the labels `LAW`, `CASELAW`, and `LITERATURE` are chosen for the classification task. An example per label is shown in Figure 32.

| Citation | Type |
|---|---|
| Art. 12 Abs. 1 BGG | Law |
| BGE 133 II 292 E. 3.2 S. 2 | Caselaw |
| AJP 3/2004, S. 343 ff. | Literature |

Figure 32: Examples for the different citation types

The distribution of the cited documents is shown previously in chapter 3.3.1.3. While the chosen labels `LAW` and `CASELAW` directly correspond to the cited documents, the label `LITERATURE` includes all other types of citations, for example citations of essays, commentaries, book reviews, or other literature. The resulting distribution of the labels is shown in Figure 33.

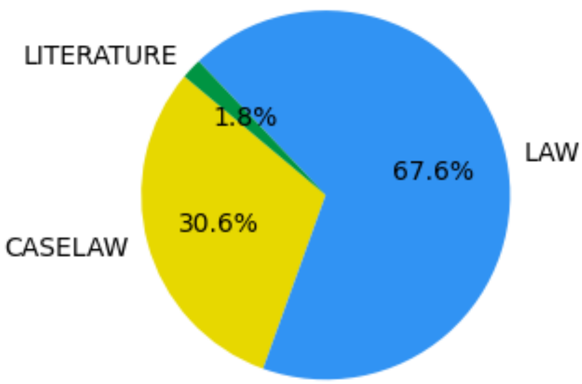



Figure 33: Distribution of citation labels in the dataset

The `LITERATURE` label is heavily underrepresented in the dataset, as it only makes up around 2% of the samples. However, it is difficult to balance the dataset in a way that would only increase the number of citations. For every chunk with literature citations, there tend to be more case law and law citations within the same chunk. Therefore, it is not possible to boost the number of literature citations without also increasing the number of other citations. Since the context of the surrounding text is crucial for the model to make accurate predictions, it is also not possible to only aggregate sentences with literature citations.

Due to this imbalance, the model might struggle to learn the features of literature citations. It is necessary to evaluate the performance of the model on the different citation types separately, and consider training separate models for each label.

### 6.1.2 Datasets

To train and evaluate the models, several datasets are utilized. This section outlines the dataset creation process, the challenges encountered, and provides an overview of the datasets themselves.

#### 6.1.2.1 Available Data

From the current regex-based solution, a large amount of data of recognized citations is available. The data is extracted from the Swisslex database and consists of documents and their citations. The data is cleaned and preprocessed to generate the datasets needed for training and evaluation. The data extraction pipeline is illustrated in chapter 6.1.2.3. Only citations linking to a document available in Swisslex are included in the training data, as it is more likely that the other citations were misrecognized by the regex solution, as described in chapter 3.3.2.1.

#### 6.1.2.2 Dataset Types

All assets and the citations they contain are divided into distinct datasets to train and evaluate the models. Each sample belongs to only one dataset, either training, test, golden or special cases. For each of these datasets, an additional version is prepared in a specific format used for training GLiNER models. The following datasets are used:

**Training Dataset** This set is created by randomly shuffling the available data. For fine-tuning, a subset can be extracted, for example the first one million chunks. The randomized nature of this dataset ensures a representative distribution of the data.

**Test Dataset** This dataset is designed to test model performance on unseen data. It is used for evaluating and comparing different models. By keeping this dataset separate from the training set, it ensures an unbiased assessment of the model's generalization capabilities.

**Golden Dataset** The golden set consists of a few hand-picked documents that cover a variety of document and citation types. The dataset is additionally manually annotated. This ensures that the models can be compared against the current regex-based solution.

**Special Cases Dataset** This dataset contains hand-selected documents that represent known special cases, as described in chapter 3.3.4. It serves as a benchmark to evaluate the models' ability to handle uncommon or challenging scenarios and is additionally manually annotated.

**Label-Specific Datasets** In addition to the general datasets, specific datasets per label (`LAW`, `CASELAW`, `LITERATURE`) are generated. This is done by merging the training and test sets, separating them by label and then splitting them again into training and test sets. For each label, only the citations that belong to the respective label are included in the dataset. This allows to train and evaluate models on specific citation types.

All datasets for recognizing citations in a text and their respective sizes are listed in Table 13.

| Name | Sample Count | Size |
|---|---|---|
| slx-dataset-citations-train.jsonl | 16′635′633 | 21.0 GB |
| slx-dataset-citations-test.jsonl] | 4′158′909 | 5.2 GB |
| slx-dataset-citations-golden.jsonl | 3′894 | 5.3 MB |
| slx-dataset-citations-specialcase.jsonl | 128 | 179 KB |
| slx-dataset-citations-caselaw-train.jsonl | 3′224′254 | 4.4 GB |
| slx-dataset-citations-caselaw-test.jsonl | 806′064 | 1.1 GB |
| slx-dataset-citations-law-train.jsonl | 6′596′100 | 9.0 GB |
| slx-dataset-citations-law-test.jsonl | 1′649′026 | 2.3 GB |
| slx-dataset-citations-literature-train.jsonl | 326′665 | 423.0 MB |
| slx-dataset-citations-literature-test.jsonl | 81′667 | 106.0 MB |
| slx-dataset-citations-train-gliner.jsonl | 16′635′633 | 26.0 GB |
| slx-dataset-citations-test-gliner.jsonl | 4′158′909 | 6.5 GB |
| slx-dataset-citations-golden-gliner.jsonl | 3′894 | 6.6.0 MB |
| slx-dataset-citations-specialcase-gliner.jsonl | 128 | 211.0 KB |

Table 13: Datasets for recognizing citations in a text

#### 6.1.2.3 Data Extraction Pipeline

To generate the citation datasets, several steps are performed. The data extraction pipeline is illustrated in Figure 34. It consists of the following steps:

1. Export citations and assets from the database
2. Merge citations and assets into a single dataset
3. Clean the dataset by removing citation tags and adding metadata fields
4. Chunk the text into parts of max. 1000 tokens (more details in chapter 6.1.2.4), separate the chunks per language into separate files
5. Deduplicate the chunks using the MinHash algorithm (more details in chapter 6.1.2.5) and merge the resulting chunks back into a single dataset
6. Split the golden set and the special cases set from the normal dataset
7. Split the dataset into training and test sets by a ratio of 70:30
8. Create a GLiNER version of each dataset

When executing the pipeline, the generated datasets are stored in a folder with the current timestamp. Additionally, a symlink is created in a folder called `current` that points to the most recent dataset. This way, the datasets are versioned and can be easily accessed. Furthermore, a backup script is set up. It can be executed manually to copy the datasets to a NAS as backup.

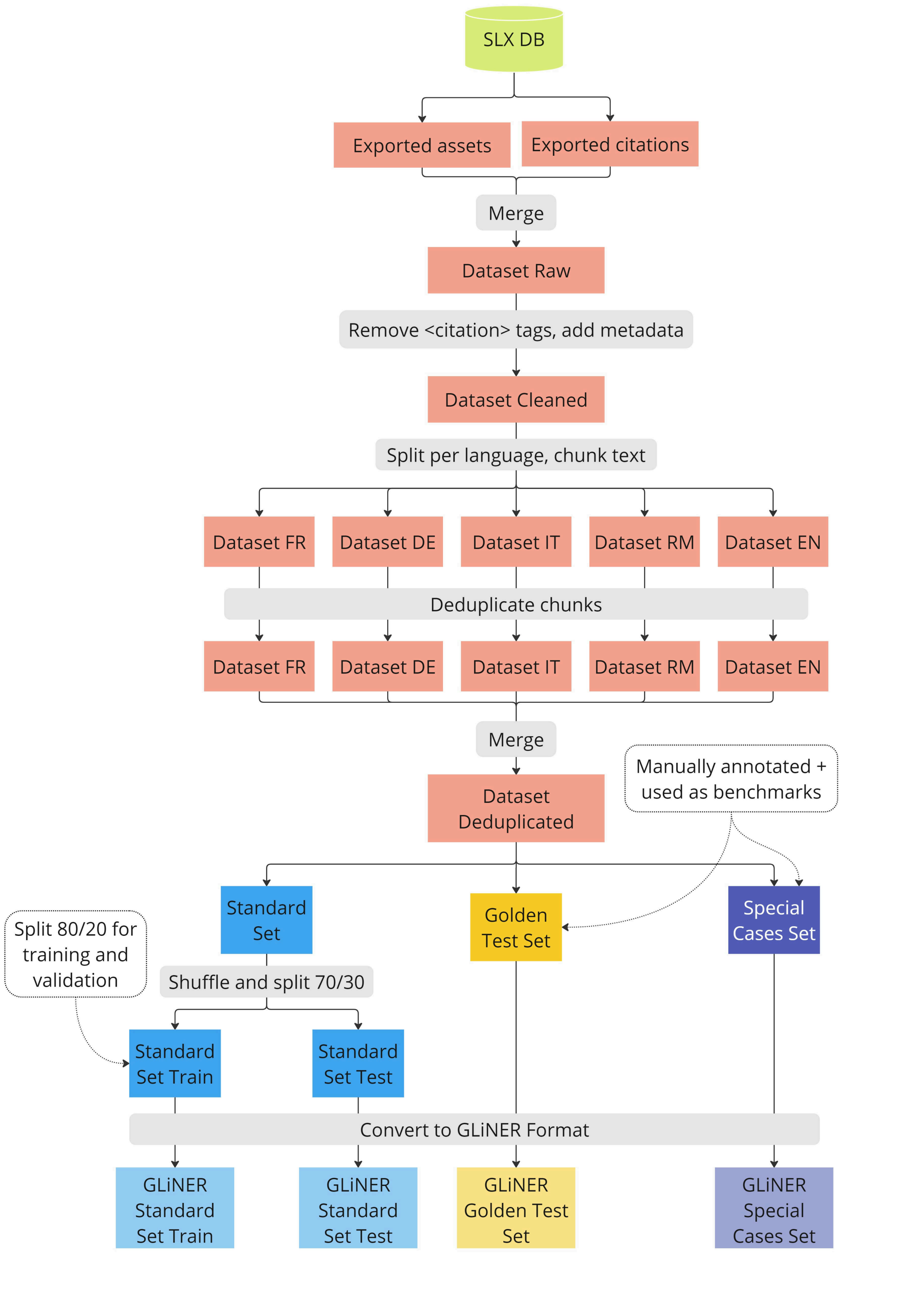



Figure 34: Data extraction pipeline

#### 6.1.2.4 Chunking

Due to the context size limitations of the models, documents are divided into smaller chunks for fine-tuning. Most BERT-based models have a context size of 512 tokens [30]. While models like Llama support significantly larger context sizes, their performance becomes slower as the context size increases [108]. For this reason, chunking is applied even for models with larger context capabilities.

A standard chunk consists of 1,000 characters. However, if a chunk boundary would fall within a known citation, the chunk is split before the citation. This ensures that citations are not cut off between chunks. Depending on the tokenizer, the number of tokens in a chunk can vary, but tests have shown that this approach is a good compromise between the number of tokens and the size of the context.

Rather than isolating individual sentences to identify citations, the entire chunk is deliberately included to provide the model with sufficient context. This allows the model to use the surrounding information to better understand whether a citation is present [109]. This is particularly important in relation to the "CO" example discussed in chapter 3.3.4.2.

During inference, the text is chunked using the same chunk size as in training to maintain consistency in the model's input structure. However, since the locations of citations are unknown during inference, a sliding window approach is employed to ensure that no citation is split across chunks. This is described in more detail in chapter 6.5.3.

#### 6.1.2.5 Deduplication

As explained in chapter 3.3.5, the dataset contains many duplicate chunks. This makes deduplication an important step to ensure the model is trained on unique data. For this purpose, the MinHash algorithm is used to sort out similar chunks during the data preproessing step. The MinHash algorithm is a probabilistic method for estimating the similarity between two sets by hashing the elements of the sets [110].

The deduplication process is implemented using the DataTrove library from Hugging Face [111] and is performed separately for each language, as the tokenizer requires specifying the language. Overall, about 12% of the chunks are removed during deduplication. Detailed statistics on the removed chunks are available in Table 14.

| Language | Total Chunks | Dropped | Percentage |
|---|---:|---:|---:|
| **German** | 15′529′361 | 1′613′696 | 10.39% |
| **French** | 7′019′348 | 1′030′991 | 14.69% |
| **Italian** | 1′034′084 | 234′746 | 22.70% |
| **English** | 97′378 | 2′759 | 2.83% |
| **Romash** | 599 | 14 | 2.34% |
| **Total** | **23′680′770** | **2′882′206** | **12.17%** |

Table 14: Deduplication statistics

#### 6.1.2.6 Sample Structure

Code-Snippet 7 shows a sample from the golden test dataset. The sample contains metadata and text. The text equals one chunk of a document, which is approximately 1000 characters. In this example, it is shortened for brevity. The identified citations are marked with spans. Each span contains the citation text, start and end position in the text, and the citation type.

```json
{
    "meta": {
        "AssetBaseId": "82E37A82-1BF7-4200-B79C-5395D5CE3A63",
        "AssetType": "Case-Law Asset",
        "Language": "Deutsch"
    },
    "text": " dass es sich beim angefochtenen Beschluss des Handelsgerichts um einen Zwischenentscheid im Sinne von Art. 92 f. BGG handelt, der das Verfahren in der Hauptsache nicht abschliesst (BGE 141 III 395 E. 2.2 V 141 E. 1.1 mit Hinweis);</p>\n<p align=\"justify\">",
    "spans": [
        {
            "id": [
                "b9002814-261e-4724-94e5-94f1a62538e4"
            ],
            "passage": "Art. 92 f. BGG",
            "start": 196,
            "end": 210,
            "label": "LAW"
        },
        {
            "id": [
                "manual"
            ],
            "passage": "BGE 141 III 395 E. 2.2 V 141 E. 1.1",
            "start": 275,
            "end": 310,
            "label": "CASELAW"
        },
    ]
}
```

Code-Snippet 7: Example sample from the golden test dataset

#### 6.1.2.7 Manually Annotated Test Sets

Since the existing test set is generated based on the data recognized by the regex, it also contains the errors made by the regex. This means that models that overcome the limitations of the regex solution are punished for it in the evaluation. Therefore, it is important to evaluate the models also on manually annotated data.

For this, two smaller test sets are manually annotated. The first test set is a random sample of the data (golden test set), while the second test set contains special cases that are not well recognized by the regex.

It is important to note that the manual labeling is a best effort annotation and not a perfect solution. The question of what constitutes a citation is not always clear. For example, if a citation is only partially visible, it is not clear whether it should be annotated or not. An example of the manual labeling tool Prodigy [92] is shown in Figure 35. When the context is considered, it is clear that the `Art. 111` belongs to the `AsylG` and that together, these two parts form a citation. However, the `Art. 111` on its own is not a usable citation, since it cannot be linked to the corresponding law.

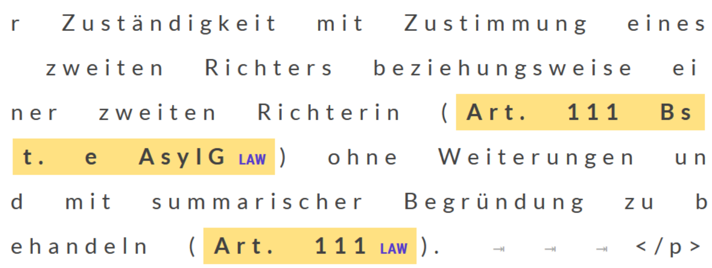



Figure 35: Example of a partial citation

Additionally, legal knowledge would be required to make better annotations. Not all abbreviations are known to the annotators. As an example in Figure 36, the abbreviation `BzP` looks like an abbreviation for a law. But in fact, it is an abbreviation for "Befragung zur Person", hence not a citation.

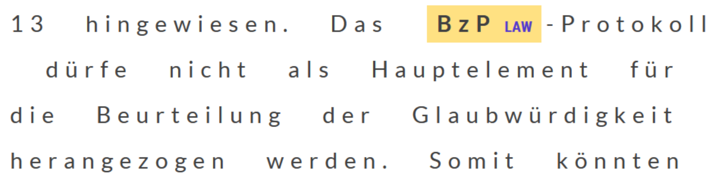



Figure 36: Example of an unknown abbreviation

Lastly, the labeling tool shows chunks of around 1000 characters at a time. Citations that are split between two chunks cannot be annotated as a single citation.

### 6.1.3 Model Selection

To find the most promising base model for recognizing citations, 21 different base models are trained and evaluated on the same dataset. Both a generative and a discriminative approach are tried. For the generative test, Llama models of different sizes are used. For the discriminative test, BERT and GLiNER models are selected. The selection of the models aims to cover a wide range of different models and sizes.

Based on the literature review in chapter 4.1.4, two Switzerland-specific models are also included in the evaluation: `ZurichNLP/swissbert` and `ZurichNLP/swissbert-ner`. They are chosen because they are fine-tuned on the languages German, French, Italian, and Romansh, which are four of the five languages present in the training set. Additionally, they were pre-trained on text related to Switzerland, which might help in understanding the legal texts.

All models are downloaded from the Hugging Face model hub. [112]

In Table 15, the models that are trained and evaluated are listed with its respective specifications. The choice of models is based on different factors such as the model's size, the language, and its case sensitivity.

| Type | Model | Params | Language | Cased |
|---|---|---|---|---|
| GLiNER | urchade/gliner_large-v2.1 | 459.0M | English | Yes |
| GLiNER | urchade/gliner_medium-v2.1 | 209.0M | English | Yes |
| GLiNER | urchade/gliner_small-v2.1 | 166.0M | English | Yes |
| GLiNER | urchade/gliner_multi-v2.1 | 209.0M | Multilingual | Yes |
| BERT | google-bert/bert-base-cased | 109.0M | English | Yes |
| BERT | google-bert/bert-base-uncased | 110.0M | English | No |
| BERT | google-bert/bert-base-multilingual-uncased | 168.0M | Multilingual | No |
| BERT | google-bert/bert-large-uncased | 336.0M | English | No |
| BERT | dslim/bert-base-NER-uncased | 109.0M | English | No |
| BERT | dslim/bert-large-NER | 334.0M | English | Yes |
| BERT | distilbert/distilbert-base-uncased | 67.0M | English | No |
| BERT | distilbert/distilbert-base-cased | 65.8M | English | Yes |
| BERT | FacebookAI/xlm-roberta-large | 561.0M | Multilingual | Yes |
| BERT | microsoft/deberta-v3-base | 86.0M | English | Yes |
| BERT | microsoft/deberta-v3-large | 304.0M | English | Yes |
| BERT | microsoft/mdeberta-v3-base | 86.0M | Multilingual | Yes |
| BERT | ZurichNLP/swissbert | 160.0M | Multilingual | Yes |
| BERT | ZurichNLP/swissbert-ner | 152.0M | Multilingual | Yes |
| Llama | meta-llama/Llama-3.2-1B-Instruct | 1.2B | Multilingual | Yes |
| Llama | meta-llama/Llama-3.2-3B-Instruct | 3.2B | Multilingual | Yes |
| Llama | unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit | 8.0B | Multilingual | Yes |

Table 15: Specifications of the evaluated models

### 6.1.4 Training Iteration

The training iteration is the process of training and evaluating a model using the datasets defined in chapter 6.1.2 as well as the `master_finetune.py` file from the Python project (chapter 5.2). The training iteration is repeated until the model reaches the desired performance or the experiment is stopped. The training iteration consists of the following steps visualized in Figure 37.

1. **Read Model Names from Config**: The model names to train and evaluate are read from a configuration file. For each model, steps 2 and 3 are executed.
2. **Fine-Tune Model**: The selected model is fine-tuned on a certain number of samples. The selected number of samples from the `train` dataset is split in a 80/20 ratio for training and validation.
3. **Evaluate Model**: The model is evaluated on the `test` data set. The number of test samples is 30% of the number of training samples (calculated before the 80/20 validation split). Precision, recall and F1 score are calculated and logged in MLflow [91].
4. **Analyze Metrics in MLflow**: The arguments and metrics can be analyzed in MLflow to determine the performance of the models.
5. **Adjust Training**: If the models do not achieve the desired performance, the training iteration is repeated with adjusted hyperparameters.

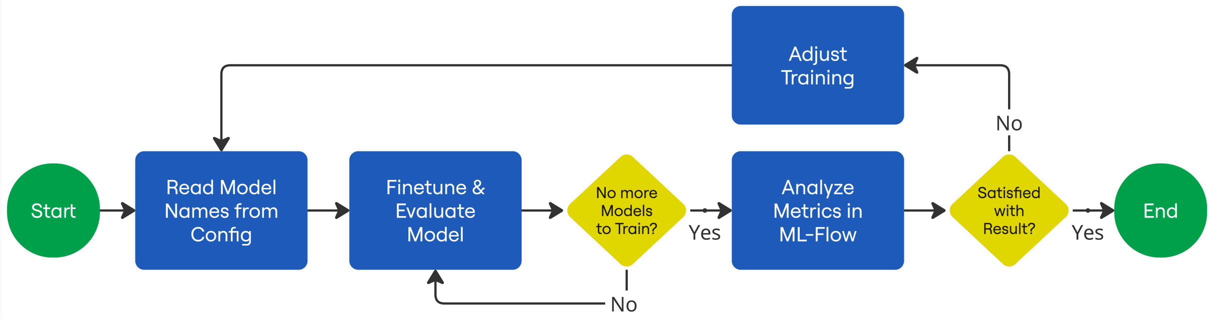



Figure 37: Training iteration

The `master_finetune.py` script simplifies the training iteration process by providing a simple interface for fine-tuning and evaluating models. There are separate scripts for training and evaluation that can be used.

### 6.1.5 Implementation Details

For fine-tuning and inference, different techniques and strategies are used. The following sections outline a few of them.

#### 6.1.5.1 LoRA

This bachelor thesis uses LoRA with its Hugging Face implementation [113] of the PEFT library [114] to fine-tune the Llama and BERT models. The implementation is based on the paper *"Low-Rank Adaptation of Large Language Models: Structured Sparse Attention and Efficient Fine-tuning"* [115].

LoRA is a technique for fine-tuning large pre-trained models by introducing small, trainable low-rank matrices. It significantly reduces the number of parameters that need updating. This method offers a key advantage in efficiency, as it requires substantially less computational power and storage compared to traditional fine-tuning approaches. This makes it particularly suitable for resource-constrained tasks. However, its effectiveness can vary depending on the complexity of the task, as the limited parameter updates may restrict its ability to capture highly intricate patterns.

It is particularly advantageous when applied to LLMs like Llama, which have billions of parameters. Updating only the small low-rank matrices enables efficient adaptation to specific tasks without requiring full model retraining. This approach not only preserves the valuable pre-trained knowledge of the model, but also facilitates task-specific customization with minimal resource usage. [115]

For GLiNER, the LoRA implementation cannot used because it requires its own trainer, which is not compatible with the SFTTrainer [116].

#### 6.1.5.2 Quantization

Quantization is a technique that reduces the precision of a model's numerical computations, such as using 8-bit integers instead of 32-bit floats, in order to decrease memory and computational requirements. It is widely used to improve the efficiency of machine learning models, making them faster and more cost-effective for large-scale or resource-constrained applications. For example the Llama model `meta-llama/Llama-3.1-8B-Instruct` uses 16.07 GB of storage for its `.safetensor` files [117]. In contrast, the quantized version `unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit` uses only 5.7 GB of storage [106].

#### 6.1.5.3 Prompt for Llama

A prompt is used to fine-tune and inference the Llama model. It is divided into three parts: system, user and assistant. Both the system and the user part stay the same while fine-tuning and inferencing. The assistant part is only used during the fine-tuning process. It stays empty during the inference process, as this is the part where the model generates the output. An example of the whole prompt can be seen in appendix section F.2.3.1.

#### 6.1.5.4 Maximal Token Length for Llama

The maximal token length for Llama determines the size of the input text chunks. To choose the right size, an experiment is conducted to analyze the token size distribution of the training data. It shows that from `100'000` training samples, the majority of the samples have a token size between `1131` and `1206`. Since there are outliers with up to `1960` tokens, a chosen token size of `2048` ensures that no input is truncated. A boxplot which displays the distribution of the token sizes can be seen in appendix section F.2.3.2.

#### 6.1.5.5 Token Classification

Token classification is a type of sequence labeling task that assigns a label to each token in a sequence. It is commonly used for NER. The datasets need to be tokenized before they can be used for training. [46], [118], [119]

When implementing NER, the labels which are used to tag tokens during classification have to be defined. For training, the tokenized dataset has to be aligned with these labels. This thesis uses the IOB2 tagging scheme which is often employed to label sequences [119]. It uses three prefixes:

- **B- (Beginning):** Marks the first token of an entity (e.g., `B-LAW` for the first token of a law citation).
- **I- (Inside):** Marks subsequent tokens inside the same entity (e.g., `I-LAW` for a part of a law citation following the first token).
- **O (Outside):** Marks tokens that are not part of any entity. In IOB2, every entity starts with a B- tag for consistency, even if it is the only token in the entity.

Prefixed with the IOB2 tags, the labels for recognizing citations (chapter 6.1.1) are:

- `B-LAW`
- `I-LAW`
- `B-CASELAW`
- `I-CASELAW`
- `B-LITERATURE`
- `I-LITERATURE`

The prefixed labels are mapped to IDs, which are used during training and inference.

Since almost every model has its own tokenizer, it is not feasible to pre-tokenize the dataset. This would result in a separate tokenized dataset for each model, which would require more storage space and increase the generation time. Therefore, the tokenization is done on-the-fly during training and evaluation.

The tokenization process during training is shown in Code-Snippet 8. The Hugging Face library [77] is used to load the tokenizer. The dataset is then loaded and tokenized. The `transform_and_tokenize_batched` function contains the logic to tokenize the data and align the IDs (representing the chosen labels) with the tokenized text.

```
tokenizer = AutoTokenizer.from_pretrained(model_name)
datasets = load_data(dataset_name, lines, splitted=True)
tokenized_datasets = datasets.map(
    lambda x: transform_and_tokenize_batched(x, tokenizer, label2id),
    batched=True
)
```

Code-Snippet 8: Simplified tokenization during training

After tokenization, the model is loaded from Hugging Face and the *labels to id* mapping and vice-versa are provided to the model, as shown in Code-Snippet 9.

```
model = AutoModelForTokenClassification.from_pretrained(
    model_name,
    num_labels=len(label_list),
    id2label=id2label,
    label2id=label2id,
)
```

Code-Snippet 9: Load model from Hugging Face and provide labels

A visualization of the tokenization and tagging of the dataset is shown in Figure 38.

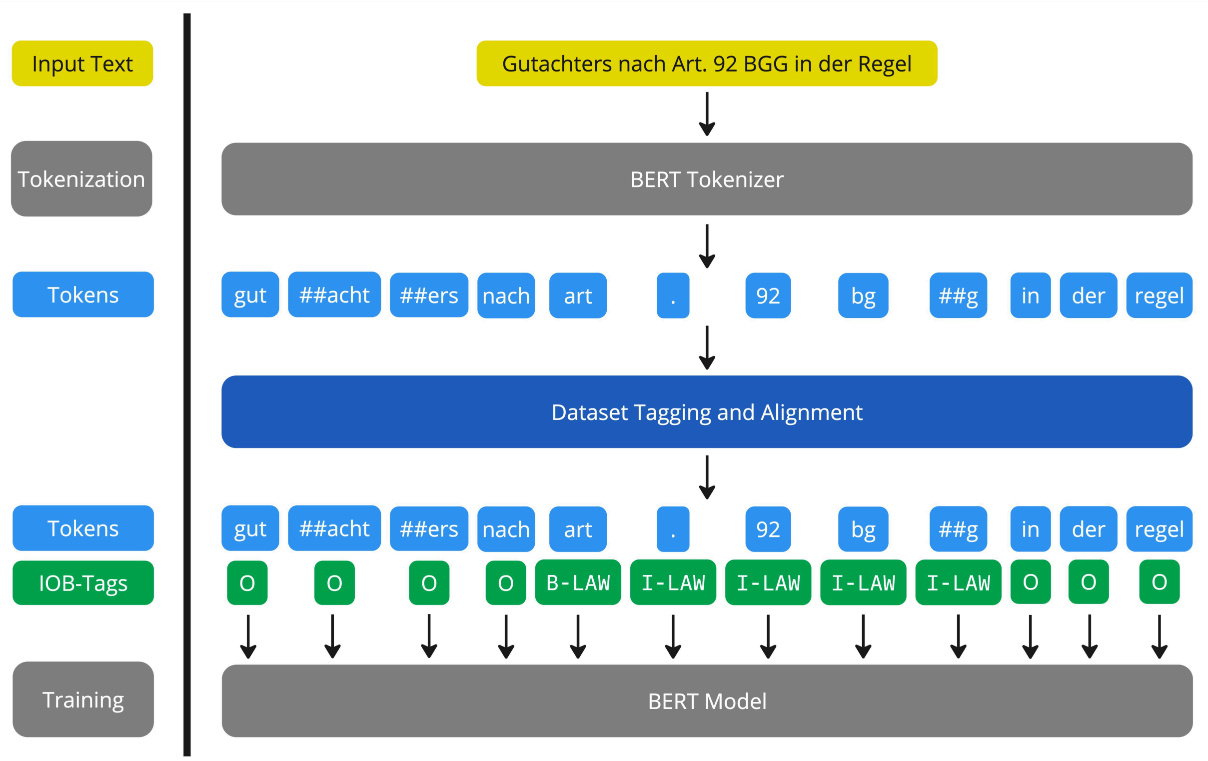



Figure 38: Tokenization and tagging of the training data

### 6.1.5.6 GLiNER Format

GLiNER operates in a span-based manner. This is in contrast to the BERT-based models described in chapter 6.1.5.5. The training data format differs from both BERT-based models and the standard dataset format described in Code-Snippet 7, requiring conversion into a seperate suitable format. This conversion is performed in the data extraction pipeline, as described in chapter 6.1.2.3.

In the pipeline, all text chunks are transformed into the `tokenized_text` format, as shown in Code-Snippet 10. Additionally, entities must be provided as spans, containing `start` and `end` indices and a `label` [38]. Due to an open GitHub issue [120], the `label` list must be explicitly included for each chunk when fine-tuning GLiNER on datasets containing chunks with an empty `ner` list.

```
{
  "tokenized_text": [
    "...",
    "Art",
    ".",
    " ",
    "423",
    "OR",
    "..."
  ],
  "ner": [
    [
      12,
      16,
      "LAW"
    ],
    ...
  ],
  "label": [
    "CASELAW",
    "LAW",
    "LITERATURE"
  ]
}
```

Code-Snippet 10: GLiNER format example sample

### 6.1.6 Hyperparameter Tuning

To find the best hyperparameters for citation recognition, a hyperparameter tuning process is conducted. Because of time constraints, the tuning is done for the DistilBERT model only, and the results are applied to the other models.

#### 6.1.6.1 Approach

Since computational resources are available, an automatic approach for hyperparameter tuning is used. To speed up the process, only a fraction of the data is used (1 million chunks) to find the best hyperparameters. The search is done using the Ray Tune library [121]. In the first step, the tested values lie within a wider range. After identifying the best hyperparameters, a second search is conducted within a narrower range around these optimal values.

#### 6.1.6.2 First Search

In Table 16, the hyperparameters used for tuning search are shown. They are based on the values discussed in chapter 4.3.2.

| Hyper-Parameter | Selection |
|---|---|
| Learning-Rate | `2e-3, 2e-4, 2e-5, 2e-6` |
| Batch-Size | `16, 32, 64, 128` |
| Weight-Decay | `0, 0.1, 0.01, 0.001` |
| Optimizer | `adamW` |
| Epochs | `1` |

Table 16: Hyperparameters used for tuning search

Out of the hyperparameters in Table 16, the best combination was found to be the following:

**Learning Rate** `0.0002` / `2e-4`
**Batch-Size** `128`
**Weight-Decay** `0.1`
**Optimizer** `adamW`
**Epochs** `1`

The experiment resulted in the following scores:

```
eval_loss: 0.005320500582456589
eval_precision: 0.9331688451700897
eval_recall: 0.9670271992801844
eval_f1: 0.9497963722594946
eval_accuracy: 0.998266342408413
```

In appendix section F.2.2, screenshots of the console output of the hyperparameter tuning process and result statistics are shown.

#### 6.1.6.3 Second Search

Based on the results of the first search, a second search is conducted with a narrower range around the best values. As scheduling method, PopulationBasedTraining is used. As with the first search, the search is done with Ray Tune on DistilBERT. Code-Snippet 11 shows the configuration of the scheduler.

```
scheduler = PopulationBasedTraining(
    time_attr="training_iteration",
    metric="eval_f1",
    mode="max",
    perturbation_interval=1,
    hyperparam_mutations={
        "weight_decay": tune.uniform(0.05, 0.2),
        "learning_rate": tune.loguniform(2e-3, 5e-5),
        "per_device_train_batch_size": tune.choice(batch_size_space),
    },
)
```

Code-Snippet 11: `PopulationBasedTraining` hyperparameter definition

The first search found the optimal batch size to be `128`. However, depending on the model size, this leads to memory problems. Therefore, the smaller values `8` and `16` are also included in the search space for the second run.

The second search results in the following scores:

**Learning Rate** `0.00010137455180990025`
**Batch-Size** `8`
**Weight-Decay** `0.19484480496118395`
**Max-Sequence-Length** `1024`
**Epochs** `1`

#### 6.1.6.4 Application to Models

Where applicable, the Llama and GLiNER models are trained with the same hyperparameters as the BERT models. However, some parameters need to be adjusted to fit the models. The final hyperparameters used can be seen in Table 17.

| Type | Value BERT | Value GLiNER | Value Llama |
|---|---|---|---|
| Epoch | 1 | 1 | 1 |
| Batch size | 8 | 16 | 8 |
| Learning rate | 0.00010137455180990025 | 2e-6 | 2e-4 |
| Weight decay | 0.19484480496118395 | 0.01 | 0.1 |
| Max sequence length | 1024 | | 1024 |
| Lora rank | 32 | - | 32 |
| Lora alpha | 64 | - | 64 |
| Lora dropout | 0.1 | - | 0 |
| Train / test split | 0.8 / 0.2 | 0.8 / 0.2 | 0.8 / 0.2 |

Table 17: Hyperparameters used for training the models

### 6.1.7 Model Evaluation

The base models shown in Table 15 are trained on 100′000 samples with a split of 80% training and 20% validation. The evaluation is done on a separate test set with 30′000 samples. The evaluation metrics are visualized in Figure 39. A detailed overview of the evaluation metrics can be found in appendix section F.3.1.1.

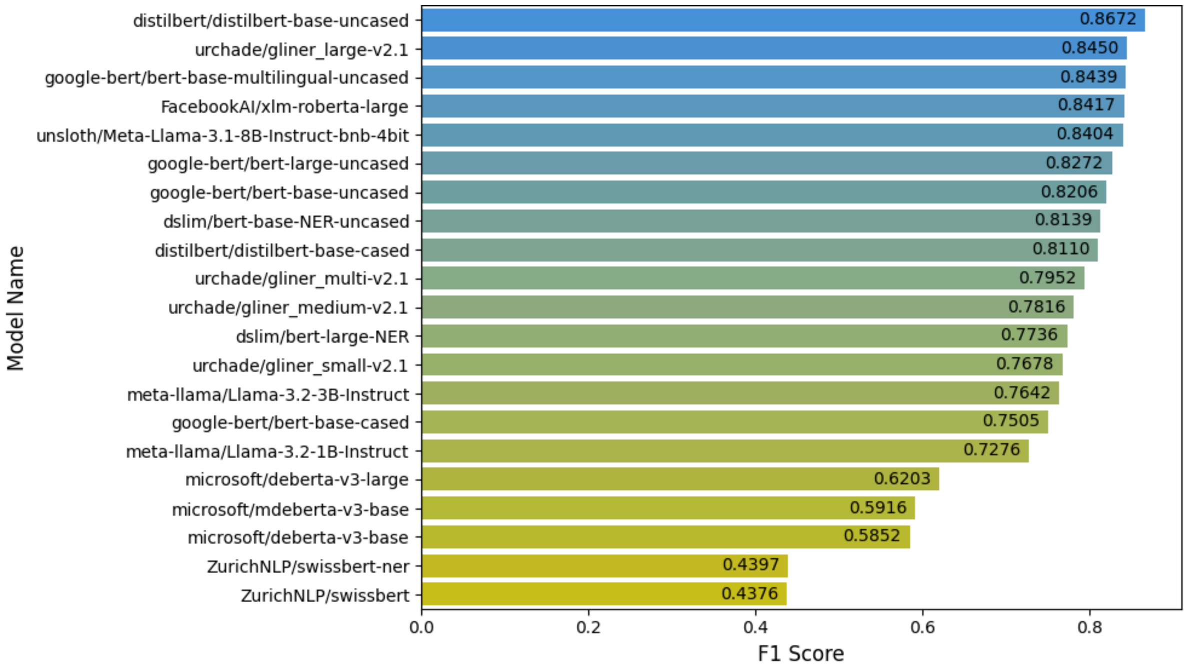



Figure 39: F1 scores of the evaluated base-models

Based on these results, multiple conclusions can be drawn.

#### 6.1.7.1 Case Sensitivity

When directly comparing two models with a cased and an uncased variant, the uncased model performs better. The cased Google BERT base model achieves an F1 score of 75.1%, while the uncased variant of the same base model achieves a significantly higher score of 82.1%. The same can be seen when comparing the two variants of the DistilBERT base model: The cased model achieves an F1 score of 81.1%, while the uncased model comes in at 86.7%.

This can be explained by the fact that cased models have to learn to distinguish between uppercase and lowercase letters, which may not be crucial for recognizing legal citations. Legal citations typically follow structured patterns, where capitalization does not alter their meaning. If a cased model overemphasizes capitalization, it might introduce unnecessary complexity and fail to generalize well. [30]

However, whether a model is cased or uncased is less significant when comparing different base models. For example, Facebook's XLM-RoBERTa large model achieves an F1 score of 84.2%. Even though it is cased, it still performes better than the comparable uncased model Google BERT large, which only reaches 82.7%.

#### 6.1.7.2 Language

The language of the model seems to have only a slight impact on the performance. When comparing the multilingual GLiNER model with its English version (GLiNER medium) with the same number of parameters, there is hardly any difference. The multilingual version reaches an F1 score of 79.5%, the English version reaches 78.2%. A similar difference is found for the Google BERT models. The Google BERT base multilingual uncased model achieves an F1 score of 84.4%, the English version gets only 82.1%. Although both are Google BERT base models, they differ in parameter size: The multilingual version has 168M, while the English version has only 110M, making a direct comparison difficult.

On the other hand, the DeBERTa models shows a slight difference in the opposite direction. The multilingual version mDeBERTa base scores 58.5%, the English version performs better with 62.0%.

Legal citations across the most common languages in the training set (German, French, English, Italian) often follow similar numeric and symbolic conventions (e.g. *Art. 2* or *§ 15*). The evaluation demonstrates that these patterns are effectively captured by the multilingual models. However, since these patterns rely less on language-specific contextual nuances, the English models perform comparably well.

#### 6.1.7.3 Size

There is no clear correlation between model size and F1 score. The best performing model (DistilBERT base uncased) has only 67M parameters, which is the second smallest of the models evaluated in terms of size. The largest model with over 8B parameters (Llama 3.1 8B) is only 5th in the ranking.

However, when comparing models from the same family, the model with more parameters usually performs better. For example, the GLiNER large model achieves an F1 score of 84.5%, while the medium model gets 78.2% and the small version scores only 76.8%. The same can be seen for the Llama models: The 8B model reaches an F1 score of 84.0%, the 3B results in 76.4%, and the smallest model with 1B parameters scores only 72.8%.

In Figure 40, the F1 score is plotted against the number of parameters of the model. Since the Llama models have a significantly higher number of parameters than the other models, they are excluded from a second plot in appendix section F.3.1.1.

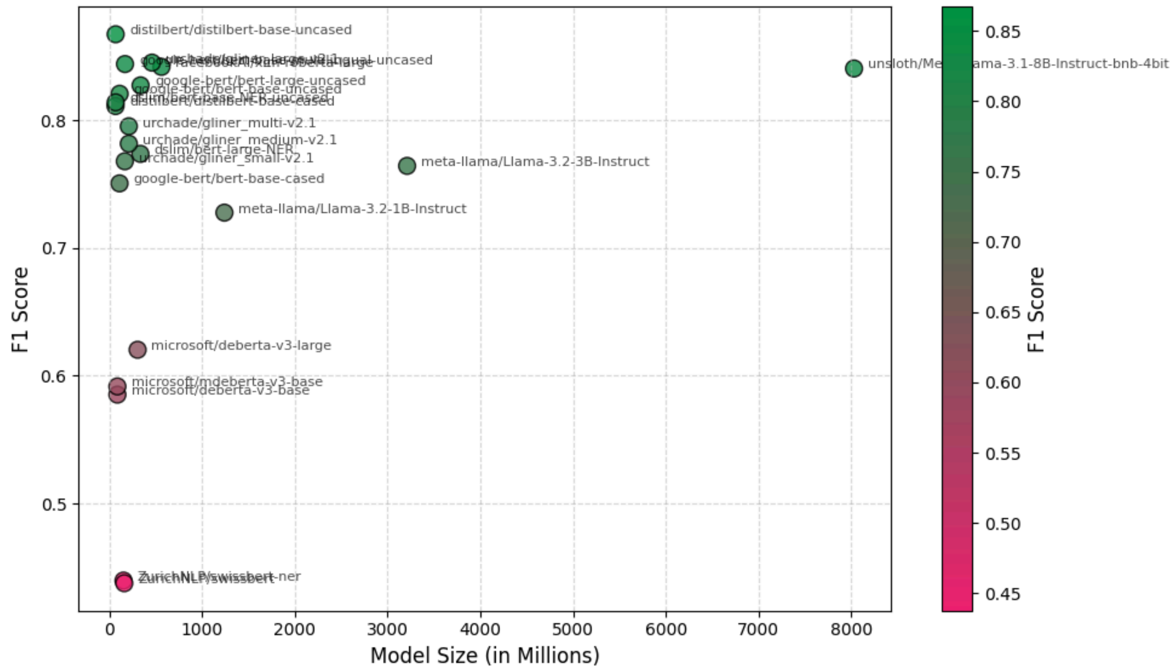



Figure 40: Correlation between model size and F1 score

For the BERT-based models, this result can be explained by how they process information. Each layer in the model learns a different type of information. The lower layers focus on simple details, like recognizing individual words and their basic roles in a sentence. The middle layers learn how words relate to each other, like understanding grammar and context. The higher layers focus on the overall meaning of the sentence, including deeper connections and context. This allows BERT models to handle both straightforward patterns and more complex relationships in the text. [122]

The Google BERT large model has 24 layers, while the base version has only 12, and DistilBERT reduces this further to 6 [30], [101]. More layers enable models to capture complex patterns, semantic relationships, and long-range dependencies in the data. However, for the task of citation recognition, long-range dependencies are less important because citations are usually short, and their parts are close together. Additionally, citations often follow structured patterns that can be effectively recognized by the lower layers of the model. This may explain why smaller models outperform larger ones for this specific task.

#### 6.1.7.4 Generative vs. Discriminative

The generative models do not perform as well as the discriminative models, especially in comparison to their model size, training time, and inference time.

This discrepancy can be explained by the fundamental differences in their architectures and objectives. Generative models, such as Llama, are typically designed to generate sequences of tokens by predicting the next token in a sequence [123]. They are more suitable for tasks like text generation, summarization, or translation. While they can handle classification tasks, these tasks are not their primary design focus. Discriminative models, on the other hand, are specifically designed for classification tasks. They predict the probability of a label given the input, which is the primary objective of NER.

#### 6.1.7.5 Precision vs. Recall

While the F1 score is a good general measure of model performance, recall is more important than precision for this project. This is because false positives (recognized citations that do not actually exist) are not a major issue; they simply do not get linked later in the process and have no significant impact. However, false negatives (missed citations that do exist) are critical, as they result in valid citations being overlooked. This would undermine the system's primary purpose of accurate citation recognition and linkage. Ensuring high recall minimizes the risk of missing these crucial citations, making it a key priority for this project.

While the recall correlates well with the F1 score, there are some outliers. All Llama models have a lower recall than their precision, which is not the case for the other models. It seems as if generative models are generally better at precision than recall, in contrast to discriminative models, such as the evaluated BERT-based models. The plot in Figure 41 shows the correlation between recall, precision and F1 score.

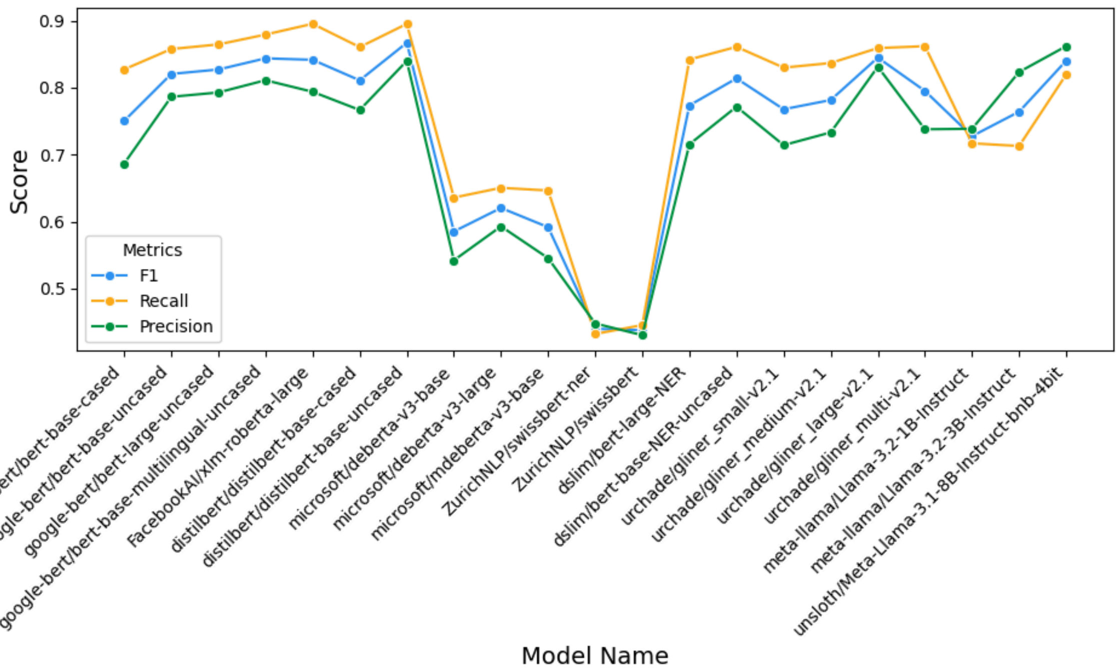



Figure 41: Correlation of precision, recall and F1 score

When ranking the models by recall, a slightly different order is obtained. The XLM-RoBERTa large model is now the best performing model, DistilBERT base uncased moved down to the second place, and the Google BERT base multilingual uncased model is in third place. The Llama models are ranked worse than in the F1 score ranking, the best one is the 8B model in 14th place. The ranking by recall is shown in Figure 42.

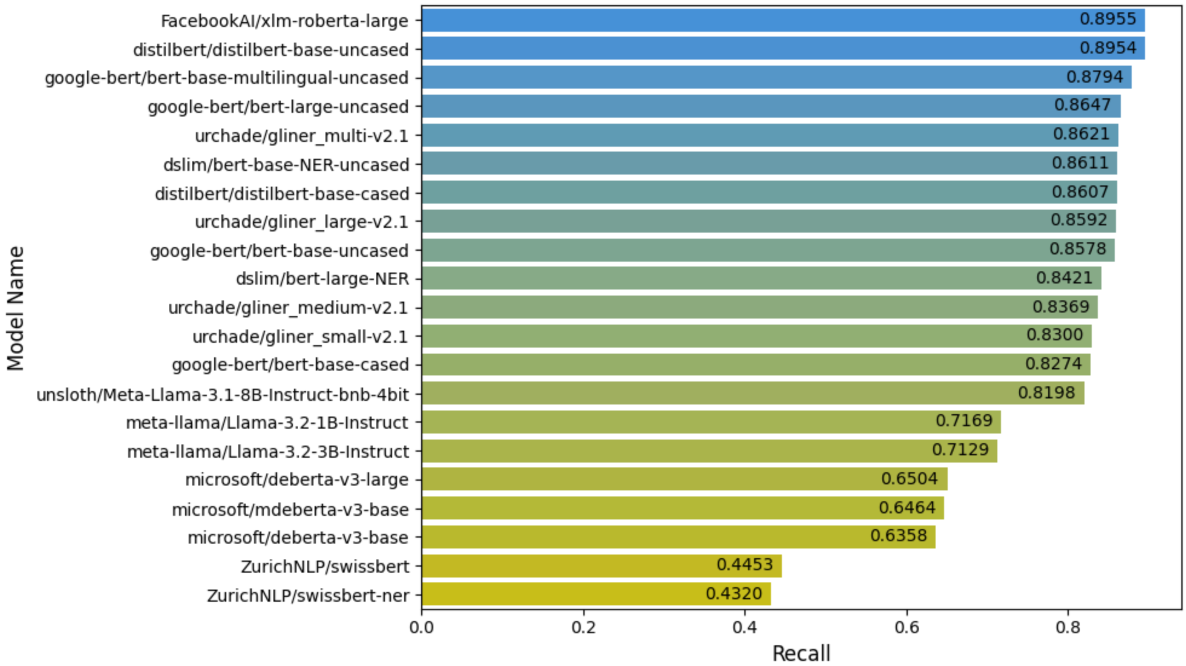



Figure 42: Recall of the evaluated base-models

### 6.1.8 Sample Size Increase

Based on the results of the evaluation in chapter 6.1.7, the models with the highest F1 scores are selected for further evaluation on a larger dataset. To analyze the influence of language when training on more data, the `urchade/gliner_multi-v2.1` model is also included in this evaluation, despite its slightly lower score.

The selected models are:

- google-bert/bert-large-uncased
- google-bert/bert-base-multilingual-uncased
- FacebookAI/xlm-roberta-large
- distilbert/distilbert-base-uncased
- urchade/gliner_large-v2.1
- urchade/gliner_multi-v2.1
- unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit

All models are trained on 1 million samples, the best performing ones in a second step on 5 and 10 million samples. The validation and test split remains the same as in the previous evaluation.

#### 6.1.8.1 Training on 1 Million Samples

The metrics of the first evaluation after training on 1′000′000 samples are shown in Table 18, together with the training time of the models. All models achieve a high F1 score, with the DistilBERT model performing best with an F1 score of 94.37%. The metrics of the evaluation are visualized in Figure 43.

| Model | Precision | Recall | F1 | Training Time |
|---|---|---|---|---|
| google-bert/bert-large-uncased | 0.9278 | 0.9516 | 0.9396 | 3.3h |
| google-bert/bert-base-multilingual-uncased | 0.9231 | 0.9540 | 0.9383 | 1.7h |
| FacebookAI/xlm-roberta-large | 0.9126 | 0.9448 | 0.9284 | 3.1h |
| distilbert/distilbert-base-uncased | 0.9349 | 0.9527 | 0.9437 | 1.4h |
| urchade/gliner_large-v2.1 | 0.9460 | 0.8587 | 0.9002 | 2.3d |
| urchade/gliner_multi-v2.1 | 0.8656 | 0.8701 | 0.8678 | 1.0d |
| unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit | 0.9529 | 0.9551 | 0.9540 | 1.4d |

Table 18: Evaluation after training on 1 million samples

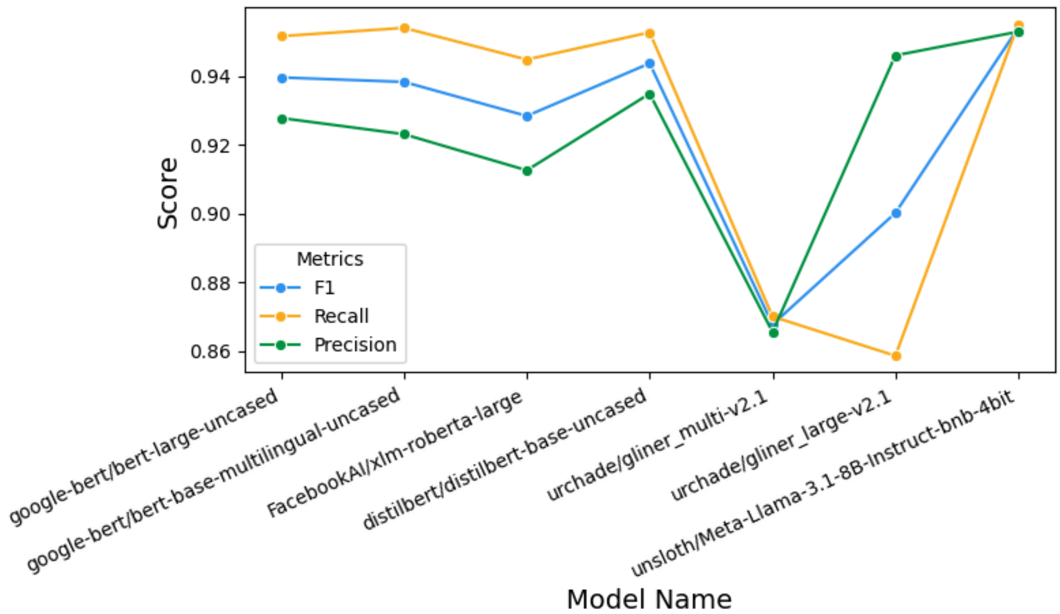



Figure 43: Evaluation after training on 1 million samples

#### 6.1.8.2 Training on 5 Million Samples

The GLiNER large model takes significantly longer to train than the other models, as seen in Table 18. The fine-tuning of the model on 1 million samples takes over 2 days. The training time of the other models is in the range of hours. The GLiNER model also has the lowest F1 score, and it is the only model where the recall is much lower than the precision. Thus, it is not taken into account for the further evaluation on 5 million samples. The metrics of the other models after training on 5 million samples are shown in Table 19.

| Model | Precision | Recall | F1 | Training Time |
|---|---|---|---|---|
| google-bert/bert-large-uncased | 0.9639 | 0.9694 | 0.9667 | 10.2h |
| google-bert/bert-base-multilingual-uncased | 0.9599 | 0.9750 | 0.9674 | 5.3h |
| FacebookAI/xlm-roberta-large | 0.9364 | 0.9543 | 0.9453 | 9.8h |
| distilbert/distilbert-base-uncased | 0.9651 | 0.9700 | 0.9676 | 4.2h |

Table 19: Evaluation after training on 5 million samples

The models show an increase in F1 score and result in almost identical scores. The best performing model is still DistilBERT with an F1 score of 96.76%. It is notable that the precision and recall of the models come very close to each other, which is a sign of a well-balanced model. The metrics of the evaluation are visualized in Figure 44.

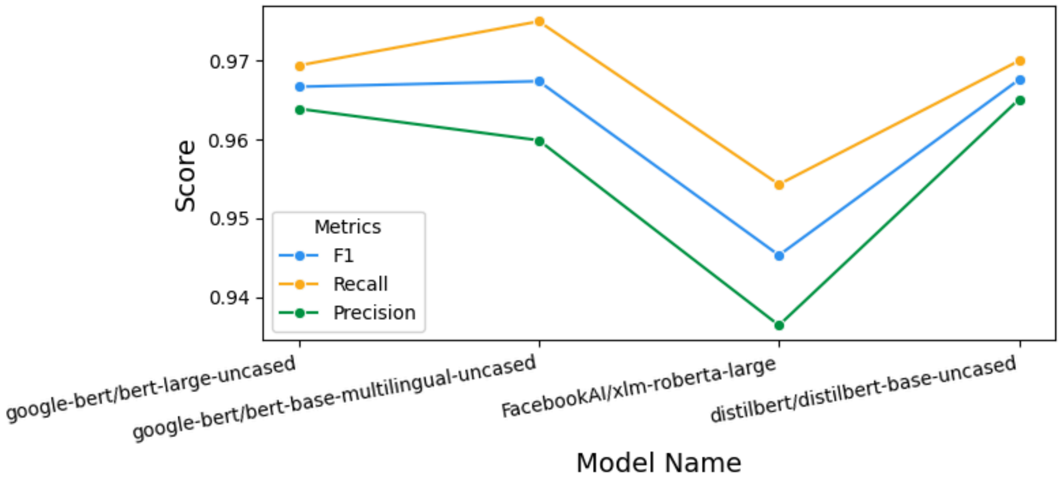



Figure 44: Evaluation after training on 5 million samples

#### 6.1.8.3 Training on 10 Million Samples

The DistilBERT and Google BERT multilingual model are additionally trained on 10 million samples. They achieve an F1 score of 96.24% respectively 95.74%. Both scores are slightly below the scores when trained on 5 million samples. However, they are still in a similar range. The metrics of the evaluation are shown in Table 20.

| Model | Precision | Recall | F1 | Training Time |
|---|---|---|---|---|
| distilbert/distilbert-base-uncased | 0.9578 | 0.9670 | 0.9624 | 14.0h |
| google-bert/bert-base-multilingual-uncased | 0.9461 | 0.9690 | 0.9574 | 17.1h |

Table 20: Evaluation after training on 10 million samples

#### 6.1.8.4 Sample Size Correlation

The results show that the F1 score increases with the number of training samples up to a certain point. However, the increase is not linear: When going from 100′000 to 1′000′000 samples, the F1 score of Google BERT large increases by 11.24%. When going from 1′000′000 to 5′000′000 samples, the increase is only 2.71%, despite the much larger increase in sample size.

Additionally, the results make clear that the impact of the choice of model decreases with the number of training samples. When trained on 100′000 samples, the F1 score of DistilBERT is 4.0% higher than that of Google BERT large uncased. When trained on 5 million samples, the difference is only 0.09%.

The correlation between the number of samples and the F1 score between 100′000 and 5 million samples is shown in Figure 45.

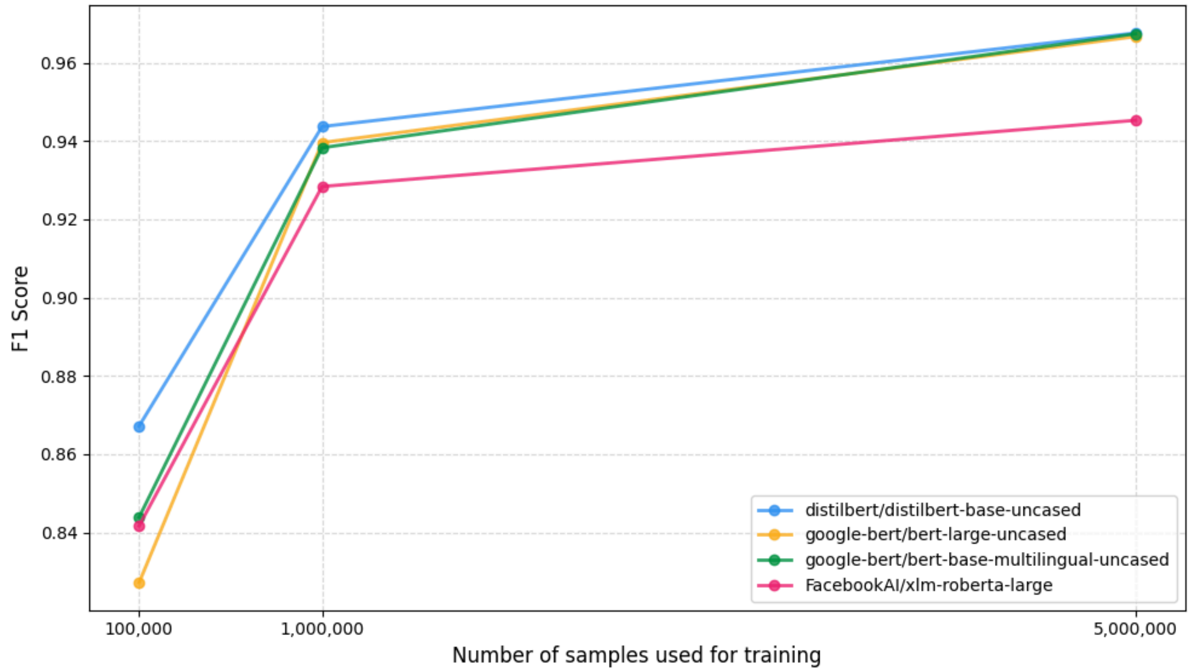



Figure 45: Correlation between training sample count and F1 score

### 6.1.9 Inference Time

The models have to process about 3000 documents per night. A document contains about 72′600 characters (chapter 3.3). It is divided into chunks of 1000 characters with a sliding window of 128 tokens (chapter 6.5.3). Since the number of characters per token varies, the number of chunks per document cannot be calculated exactly. However, it can be estimated that the average number of chunks per document is about 100 (72′600 / (1000 - 270)). This results in a total of 300′000 chunks that have to be processed per night.

To verify the speed of the models, the inference time is evaluated. For this, the time it takes the models to process a given number of chunks on the citation recognition task is measured. All models fine-tuned on 5 million samples are tested. Different sample sizes of chunks are used, namely chunk sizes of 1, 100, 1000, 5000, 10′000 and 15′000 chunks.

The results show that the inference time scales approximately linearly with the number of chunks. All models are fast enough to process the required amount of data in a reasonable amount of time. The slowest model is Google BERT large. The model takes 133.43 seconds to process 10,000 chunks. This means it would take 66.72 minutes to process 300′000 chunks. However, this is still within the time frame of one night.

Figure 46 shows the performance comparison of the models.

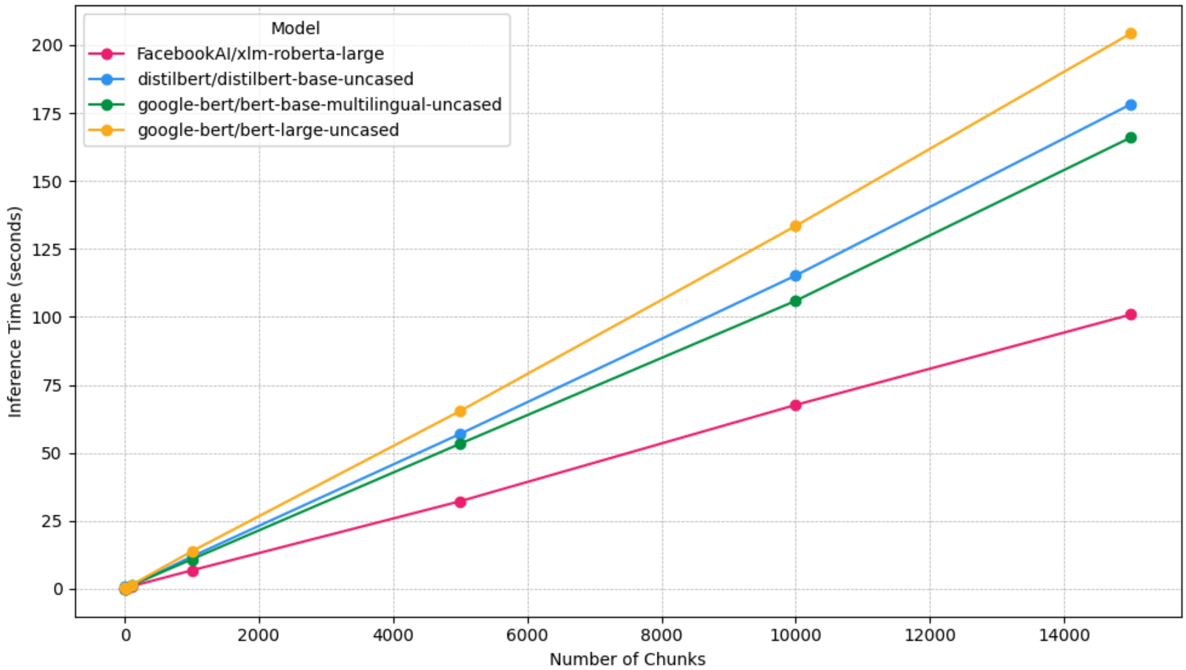



Figure 46: Performance comparison of the models

#### 6.1.9.1 Evaluation on Manually Annotated Data

As described in chapter 6.1.2.7, the models are also evaluated on manually annotated test sets to not punish models that recognize citations better than the regex solution. Both the DistilBERT and the Google BERT base multilingual uncased models are evaluated. For each model, the fine-tuned versions on 100′000, 1, 5 and 10 million samples are assessed.

When evaluated on the golden test set, the models fine-tuned on 1, 5 and 10 million samples achieve similar F1 scores of about 81%. On 100′000 samples, both models achieve an F1 score of about 77%. This is below the score achieved on the test set previously used. This shows that the models do not overfit when trained on more data, but rather generalize better.

The best performing model is the Google BERT base multilingual uncased, fine-tuned on 10 million samples. It achieves an F1 score of 81.84% and a recall value of 74.33%. The evaluation metrics for the models are shown in Table 21.

| Model | # of Samples | Precision | Recall | F1 |
|---|---|---|---|---|
| google-bert/bert-base-multilingual-uncased | 100′000 | 0.8532 | 0.7180 | 0.7798 |
| distilbert/distilbert-base-uncased | 100′000 | 0.8492 | 0.7083 | 0.7724 |
| google-bert/bert-base-multilingual-uncased | 1 million | 0.9037 | 0.7396 | 0.8135 |
| distilbert/distilbert-base-uncased | 1 million | 0.9080 | 0.7323 | 0.8107 |
| google-bert/bert-base-multilingual-uncased | 5 million | 0.9100 | 0.7406 | 0.8166 |
| distilbert/distilbert-base-uncased | 5 million | 0.9065 | 0.7369 | 0.8129 |
| google-bert/bert-base-multilingual-uncased | 10 million | 0.9103 | 0.7433 | 0.8184 |
| distilbert/distilbert-base-uncased | 10 million | 0.9025 | 0.7378 | 0.8119 |

Table 21: Evaluation metrics on the golden test set

The best performing models are also evaluated on a special cases test set. They all achieve a relatively low F1 score around 50%. This is due to the fact that the special cases are passages that are not well recognized by the regex solution, and therefore also mostly not correct in the training data, making it harder for the model to learn these cases. The best performing model is also the Google BERT base multilingual uncased, fine-tuned on 10 million samples. It achieves an F1 score of 50.89%. The evaluation metrics for the models are shown in Table 22

| Model | # of Samples | Precision | Recall | F1 |
|---|---|---|---|---|
| google-bert/bert-base-multilingual-uncased | 5 million | 0.8993 | 0.3520 | 0.5060 |
| distilbert/distilbert-base-uncased | 5 million | 0.9198 | 0.3409 | 0.4975 |
| google-bert/bert-base-multilingual-uncased | 10 million | 0.9046 | 0.3541 | 0.5089 |
| distilbert/distilbert-base-uncased | 10 million | 0.9213 | 0.3402 | 0.4969 |

Table 22: Evaluation metrics on the special cases test set

### 6.1.10 Performance per Label

In all previous evaluations, the F1 score is calculated as weighted average over all labels. However, as analyzed in chapter 6.1.1, the label `LITERATURE` is strongly underrepresented in the dataset. Therefore, the performance per label needs to be evaluated.

When breaking down the F1 score per label, it becomes apparent that the F1 score for `LITERATURE` is indeed consistently lower than for the other labels. This underperformance is not as visible in the weighted average, as the number of samples for literature citations in the test set is much smaller.

The F1 scores per label for the DistilBERT model trained on 10 million samples are shown in Figure 47. However, the same trend can be observed for all models.

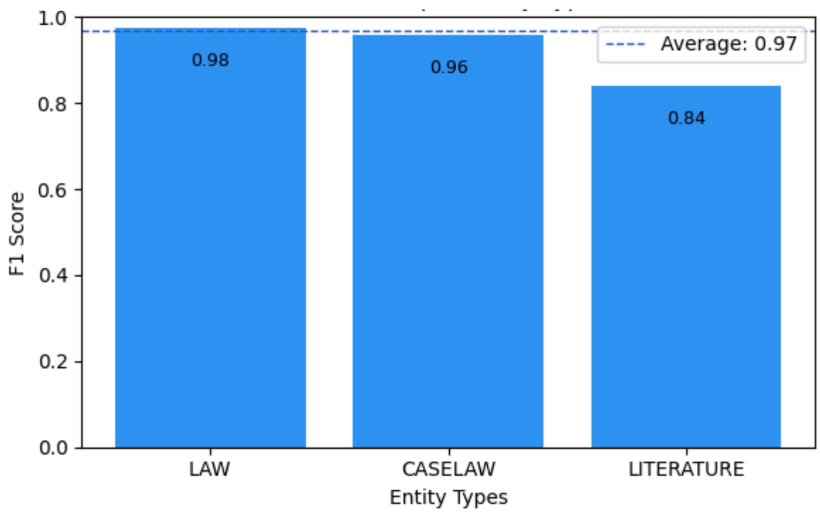



Figure 47: F1 scores per label for DistilBERT trained on 10 million samples

Due to this discrepancy and the problem that the dataset is difficult to be balanced (chapter 6.1.1), an ensemble of experts per label, as described in chapter 4.5.6, is evaluated in the next sub-chapters.

#### 6.1.10.1 Experts per Label

Since three labels are defined for citations, three experts are fine-tuned on these labels.

With this approach, each model is trained on a dataset that only contains the label it is supposed to recognize. How the datasets are created is described in chapter 6.1.2.2. Since only the chunks that contain the label are included, the model sees more examples of the label it is supposed to recognize. For the final solution, an ensemble of models would be used.

Since the `LITERATURE` training dataset contains only around 320′000 chunks, the model is trained on the entire dataset. For the `CASELAW` and `LAW` datasets, 1 million chunks and 5 million chunks are used for two training runs, respectively. The choice of models is based on the previous evaluations. Since the models are expected to perform similar on this task and due to time constraints, not all models are evaluated. The models are trained with the same hyperparameters as the models that are trained on all labels.

In Table 23, the evaluation of the literature expert models is shown. It is evident that the F1 score is higher when the model is trained only on literature citations. As seen in Figure 47, the F1 score for literature citations is at 84% when the DistilBERT model is trained on all labels. When training DistilBERT only on literature citations, the F1 score increases to 92%.

| Model | # Samples | Precision | Recall | F1 | Training Time |
|---|---|---|---|---|---|
| FacebookAI/xlm-roberta-large | 326′665 | 0.8880 | 0.9176 | 0.9026 | 59m |
| distilbert/distilbert-base-uncased | 326′665 | 0.9136 | 0.9353 | 0.9243 | 25m |

Table 23: Evaluation of the literature expert models

For the `CASELAW` and `LAW` labels, the same approach is taken. However, since the DistilBERT model trained on all three labels is already performing really well on these labels, the difference is nearly negligible. Trained on all three labels, the F1 score for `CASELAW` is at 96%. When trained only on this label, the highest F1 score is at 97%. For the `LAW` label, the F1 score is at 98% for both the model trained on all labels as well as for the `LAW` expert.

The evaluation of the `CASELAW` expert models is shown in Table 24 and the evaluation of the `LAW` expert models is shown in Table 25.

| Model | # Samples | Precision | Recall | F1 | Training Time |
|---|---|---|---|---|---|
| FacebookAI/xlm-roberta-large | 1 million | 0.9099 | 0.9237 | 0.9167 | 3.0h |
| google-bert/bert-large-uncased | 1 million | 0.9526 | 0.9660 | 0.9592 | 3.1h |
| google-bert/bert-base-multilingual-uncased | 1 million | 0.9479 | 0.9710 | 0.9593 | 1.6h |
| distilbert/distilbert-base-uncased | 1 million | 0.9545 | 0.9678 | 0.9611 | 1.2h |
| distilbert/distilbert-base-uncased | 5 million | 0.9646 | 0.9704 | 0.9675 | 6.1h |

Table 24: Evaluation of the `CASELAW` expert models

| Model | # Samples | Precision | Recall | F1 | Training Time |
|---|---|---|---|---|---|
| distilbert/distilbert-base-uncased | 1 million | 0.9540 | 0.9679 | 0.9609 | 1.3h |
| distilbert/distilbert-base-uncased | 5 million | 0.9831 | 0.9773 | 0.9802 | 8.4h |

Table 25: Evaluation of the law expert models

#### 6.1.10.2 Evaluation of the Ensemble

To evaluate the experts together on the normal test set, the experts' predictions are combined. This evaluation is not completely unbiased: The training sets for the experts may contain some of the samples from the normal test set, since these are not explicitly excluded from the experts training sets. This could artificially boost the performance of the experts, as they may have seen some of the samples during training. However, the evaluation is still useful to get an insight into the potential performance of the ensemble.

The overall F1 score is at 89%. Interestingly, the score for the `LITERATURE` label is very low at 31%, as the plot in Figure 48 shows. When looking at the evaluation matrix in Figure 49, it becomes apparent that there is an exceptionally high number of true positives, especially for the `LITERATURE` label. This can be explained by the fact that the training set only contains chunks where the label is present. This could lead to a model that always predicts the label because it has never seen a chunk without the label. The same problem is observed for the labels `CASELAW` and `LAW`, but to a lesser extent.

The high F1 score achieved during the initial evaluation of the label experts in chapter 6.1.10.1 can be explained by the fact that the label-specific test set also only contains chunks with the respective label. This is not the case for the normal test set used in the ensemble evaluation.

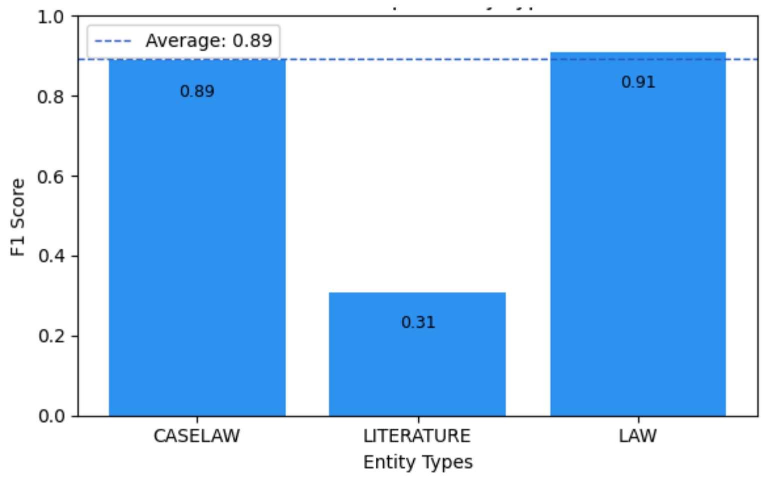



Figure 48: F1 scores per label for the ensemble of experts

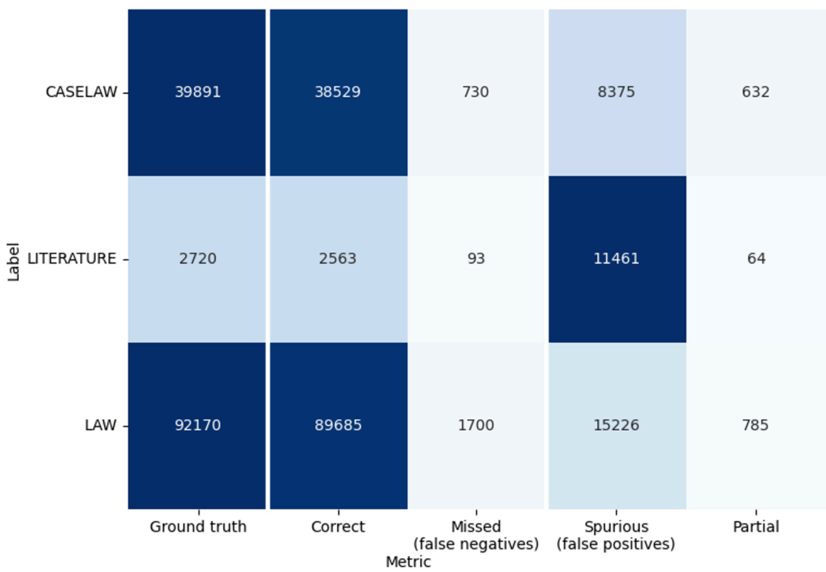



Figure 49: Evaluation matrix for the ensemble of experts

#### 6.1.10.3 Conclusion

Training models only on a specific label shows promising results. While the performance for the labels `CASELAW` and `LAW` is not significantly improved, the F1 score for the label `LITERATURE` benefits from this approach.

However, the expert ensemble does not perform as well as expected. The high number of false positives indicates that the model is biased towards predicting this label. This is probably due to the fact that the training set only contains chunks with this label.

The models could be improved by adding negative samples to the training set. This would allow the model to learn when a label is not present in a chunk. However, due to time constraints, this approach is not pursued further in this project, but mentioned as a recommendation for future work in chapter 8.3.

### 6.1.11 Post Processing

To post process the output of the models, several steps are taken to improve the quality of the results and to handle the sliding window approach. The following steps are performed:

1. **Format Responses**: The responses are formatted into a unified format, making it easier to work with the model output.
2. **Start and End Positions**: Since the Llama model does not provide start and end positions in the text, the returned passage is searched in the text and the found start and end positions are stored. This also filters out hallucinated citations that are not present in the text.
3. **Extend Citations**: All citations are extended to the next whitespace or special character in both directions. This is possible since a valid citation never stops in the middle of a word.
4. **Remove Duplicates**: Duplicate citations with the same start and end positions are removed.
5. **Handle Overlaps**: If one citation encloses another, the inner citation is removed. If two citations overlap, the citation with the higher confidence is kept, the length is extended to the maximum length, and the confidence is averaged.

These steps are applied during inference, not for the evaluation of the models.

### 6.1.12 Final Choice

Based on the results of the various evaluations, the model **`google-bert/bert-base-multilingual-uncased`** is chosen as the best model for the task of recognizing citations.

On the normal test set, it achieves one of the highest F1 scores. When compared against the manually annotated test sets, it performs the best. This shows that the model is able to generalize well to unseen data. It is well balanced in terms of precision and recall. The model is also relatively fast to train and uses fewer resources than the other models. It is able to process the required 3000 documents per night in under an hour.

The best performing version is trained on 10 million samples, which results in an F1 test score of 95.74% over all labels. This model is used for recognizing citations in the next steps of the project.

## 6.2 Recognizing Parts of a Citation

Recognizing the different parts of a citation is the second main tasks of this bachelor thesis. The data exploration for citation parts (chapter 3.3.3) has shown that there is not enough training data available for parts of legal citations, and that the existing training data is of poor quality. For judgments and journals, training data could be extracted by reverse engineering the regex solution. Apart from the fact that this is almost impossible, as described in more detail in chapter 3.3.3, the labels for the parts are poorly chosen. They are not concise enough to fulfill the given task, which is to recognize the parts of a citation *as precisely as possible* (chapter 1.3.2). Therefore, new part types are chosen, which are described in chapter 6.2.1.

Few-shot prompting with an LLM to recognize parts is evaluated. This approach does not rely on training data, but on a few examples for each citation label. It performs very well in the evaluation, but lacks the speed to process 3000 documents overnight (chapter 1.3.3). Therefore, the few-shot prompting is used as a knowledge distillation method to train a BERT-based model. This is visualized in Figure 50. Discriminative models are a lot faster. Thus, they can be used to actually process the documents in production.

The following sections describe which part types are selected, the few-shot prompting, the BERT-based approach, and the evaluation of both.

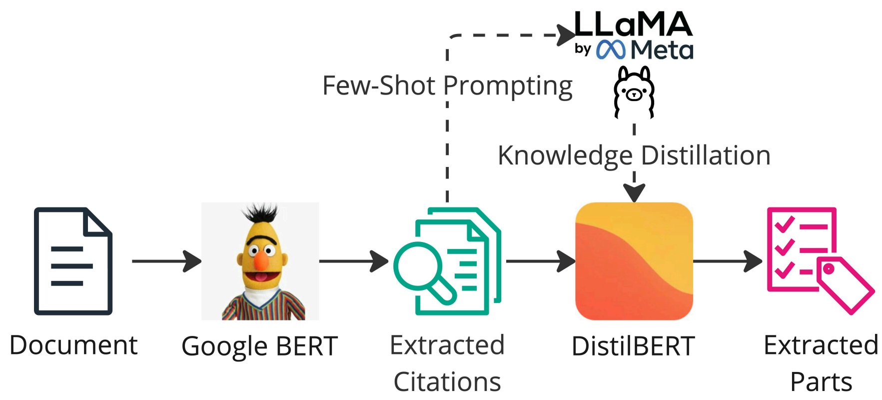



Figure 50: Processing pipeline of the citation recognition system with model chaining

### 6.2.1 Label Selection

The various citation styles require a granular classification scheme to capture the different parts of a citation. The chosen part types allow to distinguish the different parts of a citation with high precision.

Based on the citation rules (chapter 3.2) from the Swiss Federal Court [5], the types of citation parts are defined in Table 26.

| Part Type | Description | Example |
|---|---|---|
| `VOLUME` | The volume number of a citation | BGE **133** II 292 *or* SJZ **99** (2003) |
| `COURT` | The court name or abbreviation | BGE, ATF, DTF |
| `SENTENCE` | The sentence part (*Satz*) of a law citation | Satz 3 |
| `PUBLISHER` | The name or abbreviation of a journal | AJP *or* Aktuelle Juristische Praxis |
| `AUTHOR` | Authors of a journal article | MARKUS REICH |
| `PART` | The part number of a citation | BGE 133 II **292** |
| `HALFSENTENCE` | The halfsentence part (*Halbsatz*) of a citation | Halbs. **1** |
| `NUMBER` | The number (*Ziffer*) of a law citation | Ziff. **1** |
| `SUBVOLUME` | The subvolume number of a citation | BGE 133 **II** 292 |
| `DATE` | The date of a citation | Urteil des …-gerichts vom **13. November 2007** |
| `CONSIDERATION` | The consideration (*Erwägung*) part of a law citation | E. **4.2** *or* consid. **3** |
| `YEAR` | The year of a journal citation | SJZ 99 (**2003**) |
| `DECREEFORM` | The decreeform of a law citation | Bundesgesetz |
| `PAGE` | The page number of a citation | S. **123** *or* p. **123** |
| `ARTICLE` | The article of a law citation | Art. **42** |
| `LAW` | The name or abbreviation of the law | ZGB, OR |
| `REFERENCE` | The SR number of a law | SR 642.11 |
| `PARAGRAPH` | The paragraph (*Absatz*) of a law citation | Abs. **2** |
| `TITLE` | The title of a journal article | - |
| `LETTER` | The letter of a law citation | lit. **d** |
| `DOSSIERNUMBER` | The dossier number of a judgment | ATA/1243/2023 |

Table 26: Citation part types

The part types can occur in one or multiple citation labels. For example, the `PAGE` type can be found in all citation labels, whereas the `COURT` part type is only relevant for `CASELAW` citations. It is possible that the same part label occurs multiple times within the same citation. For example, a citation to a law may contain multiple `ARTICLE` part types.

### 6.2.2 Few-Shot Prompting

This chapter describes the approach to use few-shot prompting to recognize the parts of a citation. It is evaluated with two different Llama models from Meta that are served with vLLM [124].

The following models are evaluated:

- `unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit` [106]
- `unsloth/Llama-3.3-70B-Instruct-bnb-4bit` [107]

The following sections describe the used prompts, the procedure, the results and time measurements.

#### 6.2.2.1 Serving with vLLM

The vLLM package [124] is used to serve the models. The command in Code-Snippet 12 shows how to start an instance. The vLLM API is compatible to the OpenAI API, so the model can be accessed via the OpenAI client [125].

```
vllm serve unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit
    --max-num-batched-tokens 16192
    --max-model-len 16192
    --load-format bitsandbytes
    --quantization bitsandbytes
    --port PORT
    --served-model-name slx
    --enable-prefix-caching
    --enable-chunked-prefill
    --disable-log-requests
    --api-key API_KEY
```

Code-Snippet 12: vLLM serve command

The following arguments are set, because they should improve the speed of the model:

**`--quantization / --load-format`** Loading the 4 bit quantized model from unsloth reduces the model size (chapter 6.1.5.2).

**`--enable-prefix-caching`** Automatic Prefix Caching (APC) is a feature in vLLM that reuses parts of previously computed information to speed up processing [126]. Since the beginning of the prompt is always the same, it is enabled.

**`--enable-chunked-prefill`** Chunked prefill in vLLM is an experimental feature that breaks large prefill requests into smaller chunks and efficiently batches them with decode requests. It should improve GPU utilization and inter-token latency (ITL). [127].

**`--disable-log-requests`** This flag disables logging of requests [128].

#### 6.2.2.2 Prompts

A few-shot prompt is provided to the LLM to help it recognize the parts of a citation. All prompt templates are listed in appendix section F.2.5.1. The prompt consists of two parts:

**System Prompt** The system prompt is the same for all citation labels and contains the general task description and the JSON schema that the output must follow.

**User Prompt** The user prompt is specific to one citation label. It contains a general task description, the rules of the given citation label, and the citations to recognize the parts from. Each rule prompt is tailored to a specific citation label (`CASELAW`, `LAW`, `LITERATURE`) and contains examples and explanations for recognizing the parts.

The prompts are designed to be as short as possible, but as long as necessary to minimize inference time. The prompts are stored in a resource file in the *Citation Core* .NET library.

#### 6.2.2.3 Procedure

The call to the model is implemented in the *Citation Core* .NET library, since it is used in both the *Citation Checker Web* and the *Citation Processor*. The `CitationRecognitionService` class contains the logic for the recognition. The `BatchAndRecognizePartsAsync()` method is responsible for batch processing the citations and recognizing the parts. A simplified version of this method is shown in Code-Snippet 13. It is called with a list of citations and the batch size as parameters. The citations are filtered by the citation labels and grouped into batches.

CitationRecognitionService.cs

```
1  public async Task BatchAndRecognizePartsAsync(
2    List<Citation> citations,
3    int citationBatchSize,
4    CancellationToken cancellationToken = default)
5    {
6      var labels = Enum
7        .GetValues(typeof(CitationLabel))
8        .Cast<CitationLabel>()
9        .ToList();
10
11     foreach (var label in labels)
12     {
13       var citationsPerLabel = citations
14         .Where(c => c.Label == label)
15         .ToList();
16
17       foreach (var batch in citationsPerLabel.Batch(citationBatchSize))
18       {
19         await RecognizePartsAsync(batch.ToList(), label, cancellationToken);
20       }
21     }
22   }
```

Code-Snippet 13: Simplified `BatchAndRecognizePartsAsync()` method

The `RecognizePartsAsync()` method on line 19 sends the citations to the vLLM instance for text generation. It also parses the output and checks the indices of the recognized parts and corrects them if necessary. The text generation is done with the `RobustGenerate()` method (appendix section F.2.5.4). The call of this method is shown in Code-Snippet 14. The method receives a `Func` called `shouldContinue`, which defines how often a request should be repeated if the result is not as expected. This is necessary because requests to the LLM may fail or the model may return more citations than provided. The implemented logic is that the request is repeated a maximum of three times or until the result is as expected.

`RecognizePartsAsync()` in `CitationRecognitionService.cs`

```
var result = await this.LlamaService.RobustGenerate(
  messages,
  grammar,
  (counter, result) =>
     counter < 3 && (result is null || result.Citations.Count != citations.Count),
  cancellationToken);
```

Code-Snippet 14: `RobustGenerate()` call

#### 6.2.2.4 Golden Test Set

For evaluation, a hand-crafted golden test set specifically designed for recognizing parts is created, since no data from Swisslex is available (chapter 3.3.3). This test set consists of 100 citations and their corresponding parts. The samples are excluded from the training data. This makes sure that the models have never seen the golden test data before. It is ensured that each part type is represented at least once in the dataset.

#### 6.2.2.5 Performance

Llama 3.3 70B and Llama 3.1 8B are evaluated on the golden parts test set. The results can be seen in Table 27. As expected, the 70B performs better than the 8B model.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit | 0.9190 | 0.9301 | 0.9246 |
| unsloth/Llama-3.3-70B-Instruct-bnb-4bit | 0.9963 | 0.9819 | 0.9891 |

Table 27: Few-shot prompting results for recognizing citation parts on the golden test set

Manual validation has also shown that few-shot prompting works very well. However, the performance depends on the size of the model. While the 70B model is very accurate, the 8B model still makes some errors. Some example results are shown in appendix section F.2.5.3.

The 8B model sometimes makes errors by including filler words or abbreviations as part of the extracted value for a citation part. For example, in the citation "BGE 148 II 556 E. 3.4.2 ff. und E. 4," the model should extract only the value `3.4.2` for the consideration part. However, it incorrectly includes the filler word `E.` and the abbreviation `ff.` in the recognized value. The Table 28 shows a side-by-side comparison of one example citation and its recognized parts by the 8B and 70B models.

| Citation Passage | Part 8B | Part Type 8B | Part 70B | Part Type 70B |
|---|---|---|---|---|
| BGE 148 II 556 E. 3.4.2 ff. und E. 4 | BGE | Court | BGE | Court |
| BGE 148 II 556 E. 3.4.2 ff. und E. 4 | 148 | Volume | 148 | Volume |
| BGE 148 II 556 E. 3.4.2 ff. und E. 4 | II | Subvolume | II | Subvolume |
| BGE 148 II 556 E. 3.4.2 ff. und E. 4 | 556 | Part | 556 | Part |
| BGE 148 II 556 E. 3.4.2 ff. und E. 4 | **E.** 3.4.2 **ff.** | Consideration | 3.4.2 | Consideration |
| BGE 148 II 556 E. 3.4.2 ff. und E. 4 | **und** | **Consideration** | - | - |
| BGE 148 II 556 E. 3.4.2 ff. und E. 4 | **E.** 4 | Consideration | 4 | Consideration |

Table 28: Comparison of part recognition by 8B and 70B model

#### 6.2.2.6 Time Measurement

The time is measured with the console application *Citation Processor* with 10 documents of the golden test dataset. It loads these 10 documents, starts the citation recognition and then does the parts recognition with the few-shot prompting to the `unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit` vLLM instance. The batch size is set to 5, which means that 5 citations are sent to the LLM per request. The time is measured using the `Stopwatch` class from the .NET library. The results are shown in Table 29.

| | Documents | Processing Time | Documents per Minute |
|---|---|---|---|
| **Required** | 3000 | 6h | 8.333 |
| **Measured** | 10 | 16.33m | 0.612 |

Table 29: Time measurement for few-shot prompting approach

The results of the experiment are shown in appendix section F.2.5.2. The best run was able to process the 10 documents in 16.33 minutes. This is not fast enough to process 3000 documents overnight. Since the few-shot prompting with the 8B model is not fast enough, a time measurement of the 70B model is not necessary, since the throughput of the 8B is twice as high as the one from the Llama 70B [129]. This indicates that the 70B model would also not be able to process the required number of documents overnight either.

### 6.2.3 BERT-Based Approach with Knowledge Distillation

Few-shot prompting is not fast enough to process the required amount of documents overnight, as discussed in chapter 6.2.2.6. Therefore, knowledge distillation is used. A BERT-based model is fine-tuned on synthetic data generated with few-shot prompting. Knowledge distillation is explained in chapter 4.2.4.

#### 6.2.3.1 Dataset Generation

The data generation is implemented as a .NET console application called *Parts Training Data Generator*.

The application takes citations from the traing dataset for recognizing citations (chapter 6.1.2.2). It then uses few-shot prompting (chapter 6.2.2) to identify the parts of the citations. For this task, the model Llama 3.3 70B is used. The model is chosen because it provides better quality results than the 8B model. With the recognized parts, a new training dataset is created and saved as JSONL file. This process is shown as a code-snippet in appendix section F.2.5.4.

In the new dataset, chunks with no recongized parts are removed. This might lead to a model that is biased towards always predicting a part due to the lack of negative samples. However, this is not a problem. All citations passed to the model should contain at least one part. If a text passage is fed to the model that does not contain a valid citation and therefore contains no parts, it has no consequence if the model predicts wrong parts.

After removing the chunks without parts, the dataset is deduplicated. In contrast to the deduplication in the citation recognition model, only exact matches are removed.

After deduplication, the dataset is split into training and test sets with an 85/15 split. The resulting files are saved as `slx-dataset-parts-train.jsonl` and `slx-dataset-parts-test.jsonl`. Together with the manually annotated golden dataset described in chapter 6.2.2.4, the following datasets result:

| **Name** | **Sample Count** | **Size** |
|---|---:|---:|
| `slx-dataset-parts-train.jsonl` | 331′613 | 111.4 MB |
| `slx-dataset-parts-test.jsonl` | 50′000 | 16.8 MB |
| `slx-dataset-parts-golden.jsonl` | 100 | 45.0 KB |

Table 30: Datasets for training and evaluation of the parts recognition model

#### 6.2.3.2 Distribution of Part Types

After data generation, the training dataset is analyzed. The distribution of the part types is visualized in Figure 51. The types `REFERENCE`, `DECREEFORM`, `AUTHOR`, `TITLE`, `HALFSENTENCE` and `SENTENCE` are underrepresented compared to the others. This imbalance could affect the performance of the model, as it is not exposed to enough samples of these types to generalize effectively.

Analysis of the original data in the citations training set shows that this is currently not a problem. The reason is that the current regex solution does not always recognize citations exactly as it is defined in the citation rules in chapter 3.2. Especially the parts `REFERENCE`, `DECREEFORM`, `AUTHOR`, `TITLE` and `HALFSENTENCE` do not occur often in the citations that are recognized. Due to this, the model for recognizing citations (chapter 6.1) does not recognize these parts either, since it is trained on the data generated by the regex solution.

This means that during inference, the model trained to recognize specific parts is seldom exposed to these types. As a result, the model's diminished performance on these types rarely poses a significant issue.

In the future, if the model which recognizes citations is improved to include these parts more often, the model which recognizes parts should be retrained with a more balanced dataset.

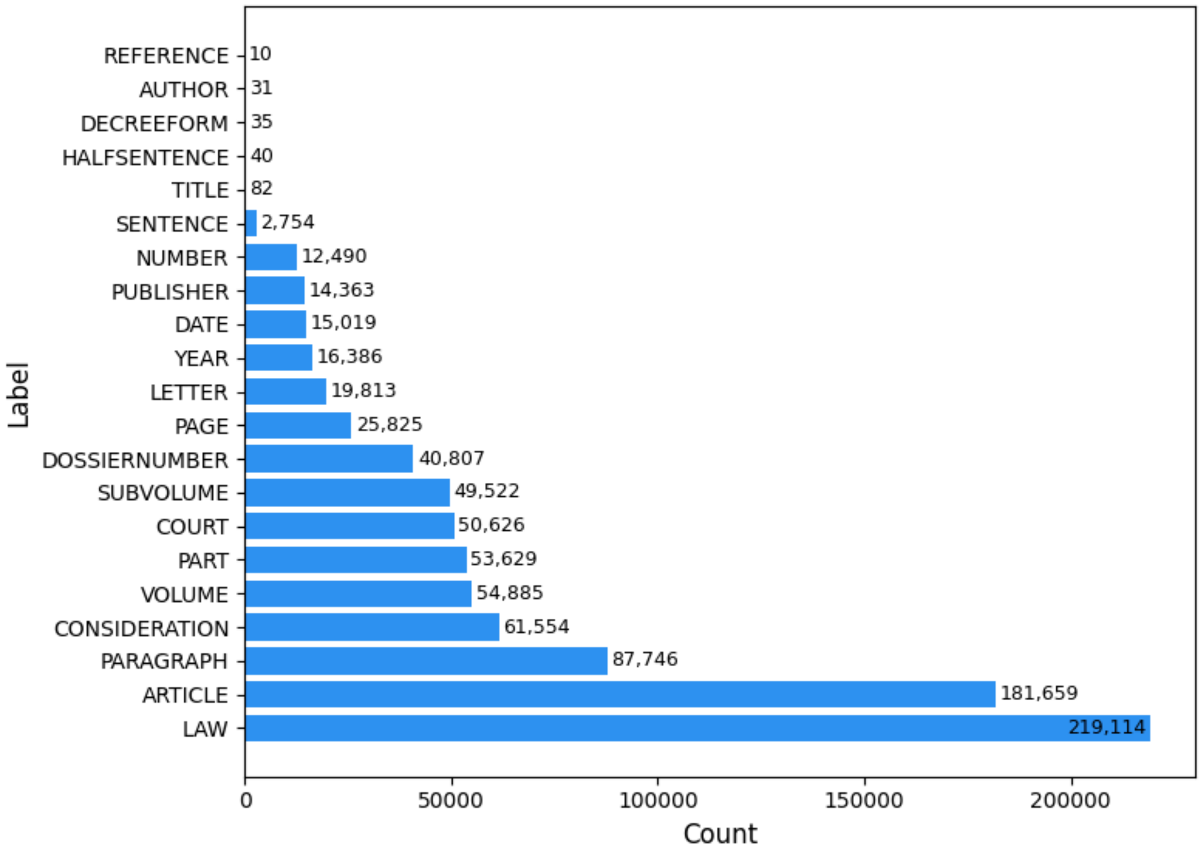



Figure 51: Distribution of the part types in the generated training data

Only the part type `SENTENCE` does not share this problem. Additional data could be generated to balance this part type. This could be done by searching the citations training dataset for text chunks containing the respective part type. Due to time constraints, this is not done in this thesis.

#### 6.2.3.3 Evaluation on Test Dataset

The base models that performed best when fine-tuned on recognizing citations are evaluated for recognizing parts. They are trained on the generated dataset for parts of citations. The training is done with the same workflow as for the fine-tuning on citations, which is described in chapter 6.1.4. Also, the same hyperparameters are used.

All models are fine-tuned on 331′613 sample citations and evaluated on 50′000 samples. This results in a train/test split of approximately 85/15. The test set is smaller than the one used for the model that recognizes citations. This is due to the limited amount of available training data. The evaluation results are shown in Table 31.

| Model | Precision | Recall | F1 | Training Time |
|---|---|---|---|---|
| distilbert/distilbert-base-uncased | 0.9876 | 0.9911 | 0.9894 | 12.1m |
| FacebookAI/xlm-roberta-large | 0.9800 | 0.9872 | 0.9836 | 30.8m |
| google-bert/bert-base-multilingual-uncased | 0.9867 | 0.9914 | 0.9891 | 17.4m |
| google-bert/bert-large-uncased | 0.9861 | 0.9922 | 0.9892 | 29.5m |

Table 31: Evaluation of the models for reocgnizing parts on the test set

The evaluation matrix for the DistilBERT model is shown in Figure 52.

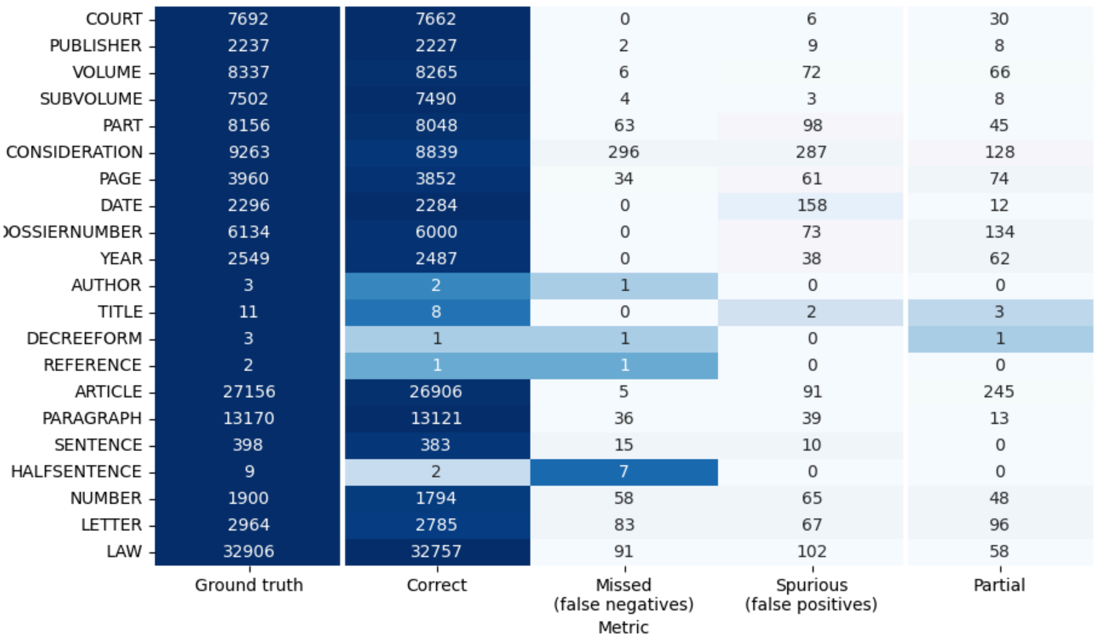



Figure 52: Evaluation matrix for recognizing citation parts test set - DistilBERT

#### 6.2.3.4 Evaluation on Manually Annotated Data

The training data for the parts recognition model is generated with few-shot prompting. The annotated spans are not manually verified. This means that the model is trained on potentially incorrect data. To verify the model's performance, the previously created golden test (chapter 6.2.2.4) set is used for evaluation. The results are shown in Table 32.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| distilbert/distilbert-base-uncased | 0.9383 | 0.9337 | 0.9360 |
| FacebookAI/xlm-roberta-large | 0.9217 | 0.9195 | 0.9206 |
| google-bert/bert-base-multilingual-uncased | 0.9620 | 0.9435 | 0.9527 |
| google-bert/bert-large-uncased | 0.9476 | 0.9361 | 0.9418 |

Table 32: Evaluation of the models on the golden test set

The evaluation matrix for the DistilBERT model is shown in Figure 53.

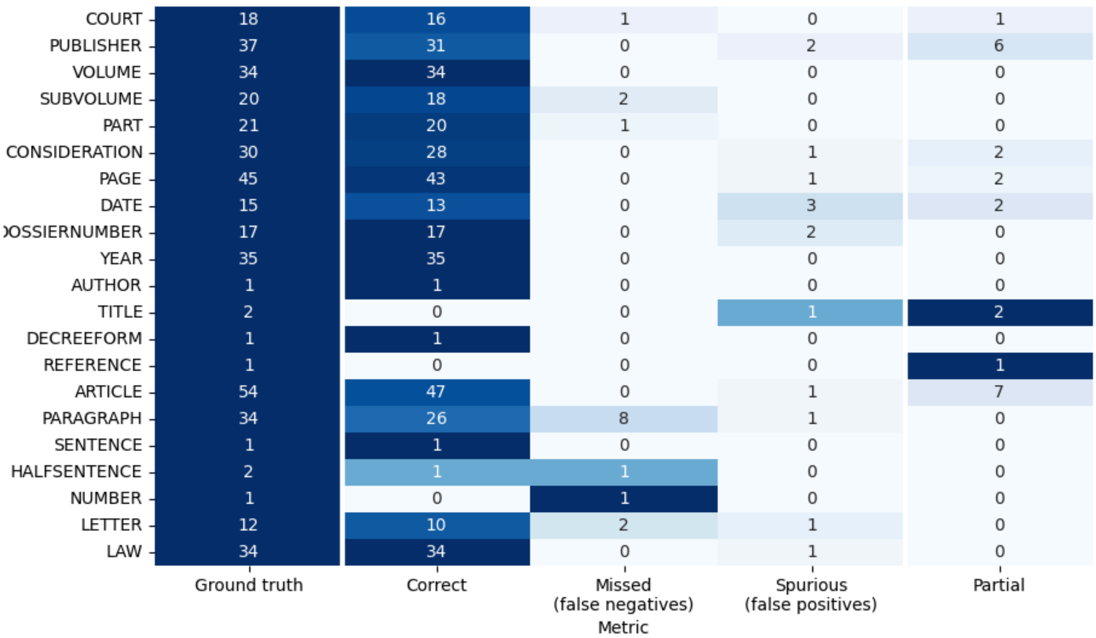



Figure 53: Evaluation matrix for recognizing citation parts golden set - DistilBERT

#### 6.2.3.5 Final Choice

In the evaluation of models for parts recognition, all tested models demonstrated exceptionally high F1 scores on the test set, ranging from 98.36% to 98.94%. When tested on manually annotated data, the performance remained strong but showed a slight decrease, with F1 scores ranging from 92.06% to 95.27%. This indicates that the data generated using few-shot prompting is not entirely accurate, but still highly effective.

Given the minimal differences in performance, the choice of model is less critical. As the final model, **`distilbert/distilbert-base-uncased`** is selected for its compact size, resource efficiency, and fast inference speed. Additionally, it retrains quickly. This could be beneficial in the future if rare part types, such as `DECREEFORM` or `REFERENCE`, are detected more frequently and require model updates (chapter 6.2.3.2). DistilBERT achieves an F1 score of 98.94% on the test set and 93.60% on manually annotated data.

Since the `google-bert/bert-base-multilingual-uncased` model slightly outperforms DistilBERT, it remains a viable alternative for the future, especially when working with a more finalized dataset.

## 6.3 Citation Checker Web

To be able to manually check the citations and parts found by the model, a web application is developed. The following sections provide an overview of the application and its components.

### 6.3.1 Architecture

The web application is built using the Blazor framework [80]. The resulting software is a single-page application that uses interactive server-side rendering, previously known as Blazor Server. The application is hosted on a Windows Server and can be accessed via a standard web browser from within the Swisslex network.

### 6.3.2 Functionality

The web application offers two starting points for the process of citation recognition:

- Uploading a text document (Figure 54)
- Searching for a Swisslex document in the database by entering its dossier number (Figure 55)

When a document is uploaded or found in the database, the text of the document is displayed. The user can then choose which models to use for the citation recognition process. Which models are available are dynamically fetched from the *Citation Inference API*.

After choosing the model that recognizes citations, the user can start the recognition process. The application sends the text to the selected model, which returns the identified citations. These are displayed in a table, together with the confidence of the model and the assigned type (`CASELAW`, `LAW`, `LITERATURE`). Also, the found citations are highlighted in the text. A short summary including the number of found citations as well as the processing time is displayed. If the text was fetched by providing a dossier number, the citations linked in the Swisslex web platform are additionally loaded and highlighted in the text.

Afterwards, a button provides the possibility to further recognize the specific parts within the citations. Again, the available models are listed in a dropdown. When starting the parts recognition, all previously found citations are sent to the inference server. The recognized parts are then marked and labeled directly in the citation in the table.

Another feature is provided to evaluate few-shot prompting on the text. This can be done either on the whole text (recognizing citations and their parts in one step, appendix section F.2.6), or only to identify the parts of the already recognized citations (chapter 6.2.2).

Errors are shown using a toast notification.

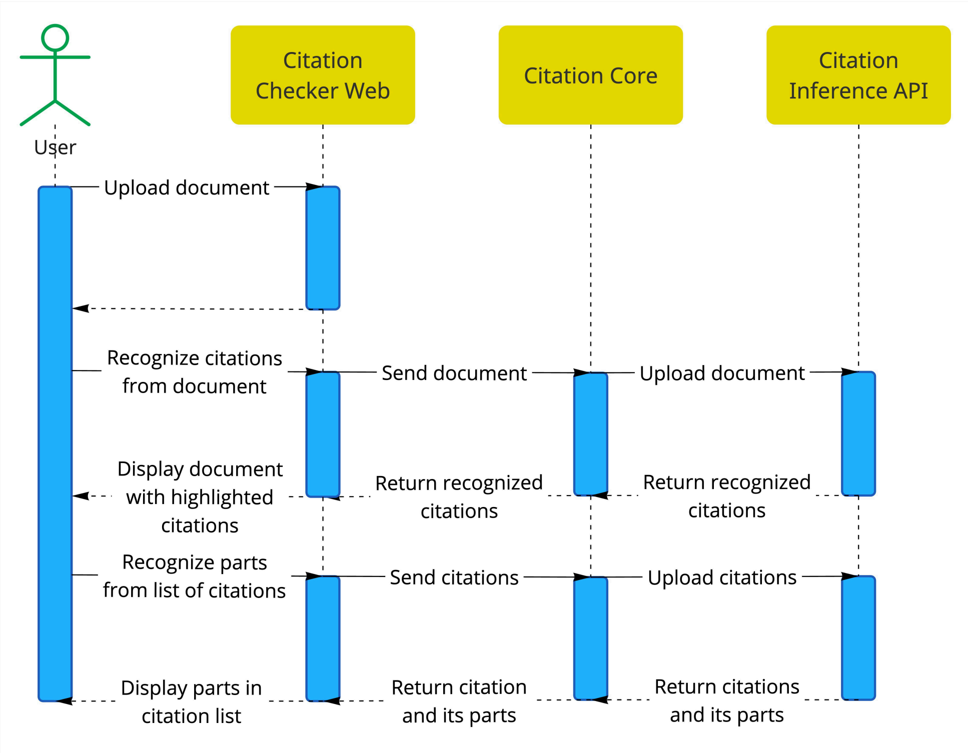



Figure 54: Sequence diagram - Citation recognition by uploading a document

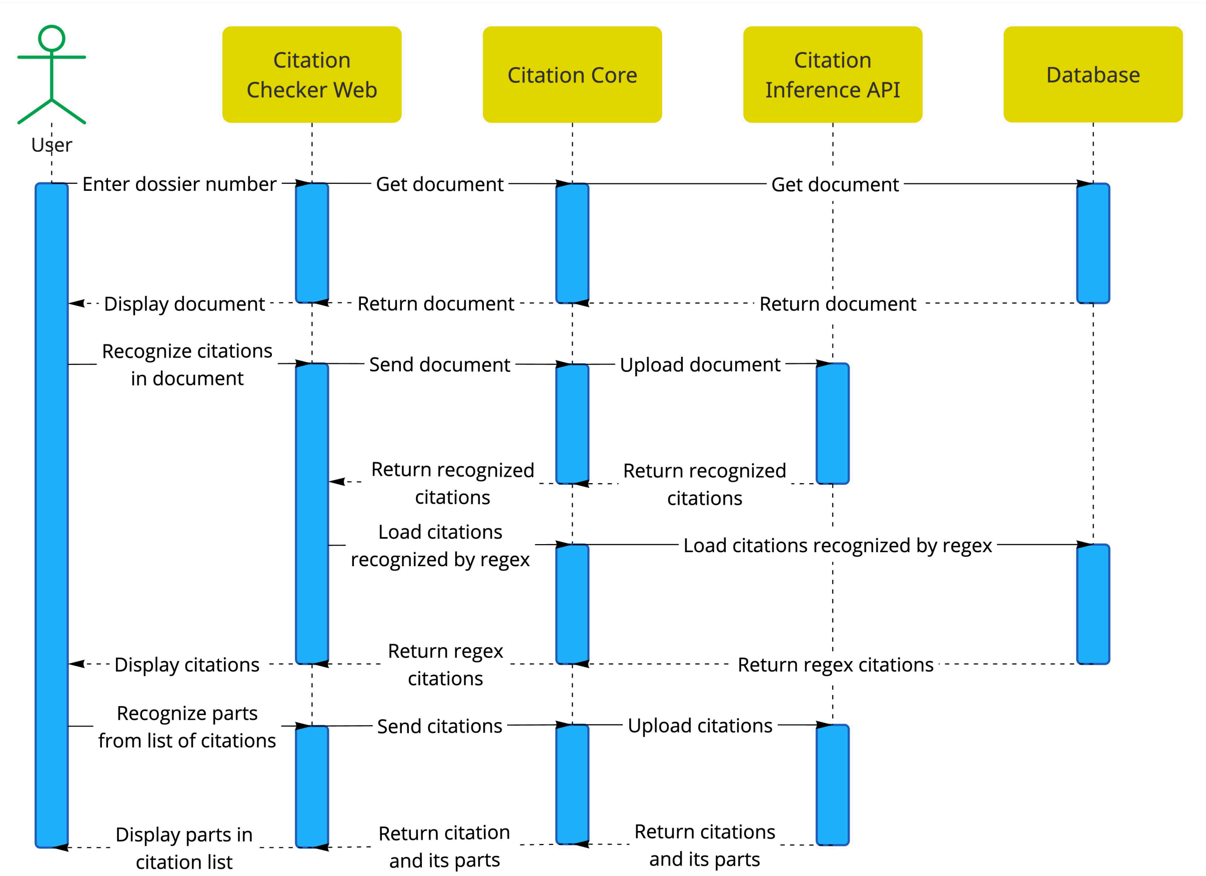



Figure 55: Sequence diagram - Citation recognition by dossier number

### 6.3.3 Design

The design is kept simple. The colors are inspired by Swisslex's corporate design. However, additional colors are used, especially for the labeling of the parts. Since the web application is only used internally for testing purposes, the design is not a priority. A screenshot of the web application can be seen in Figure 56.

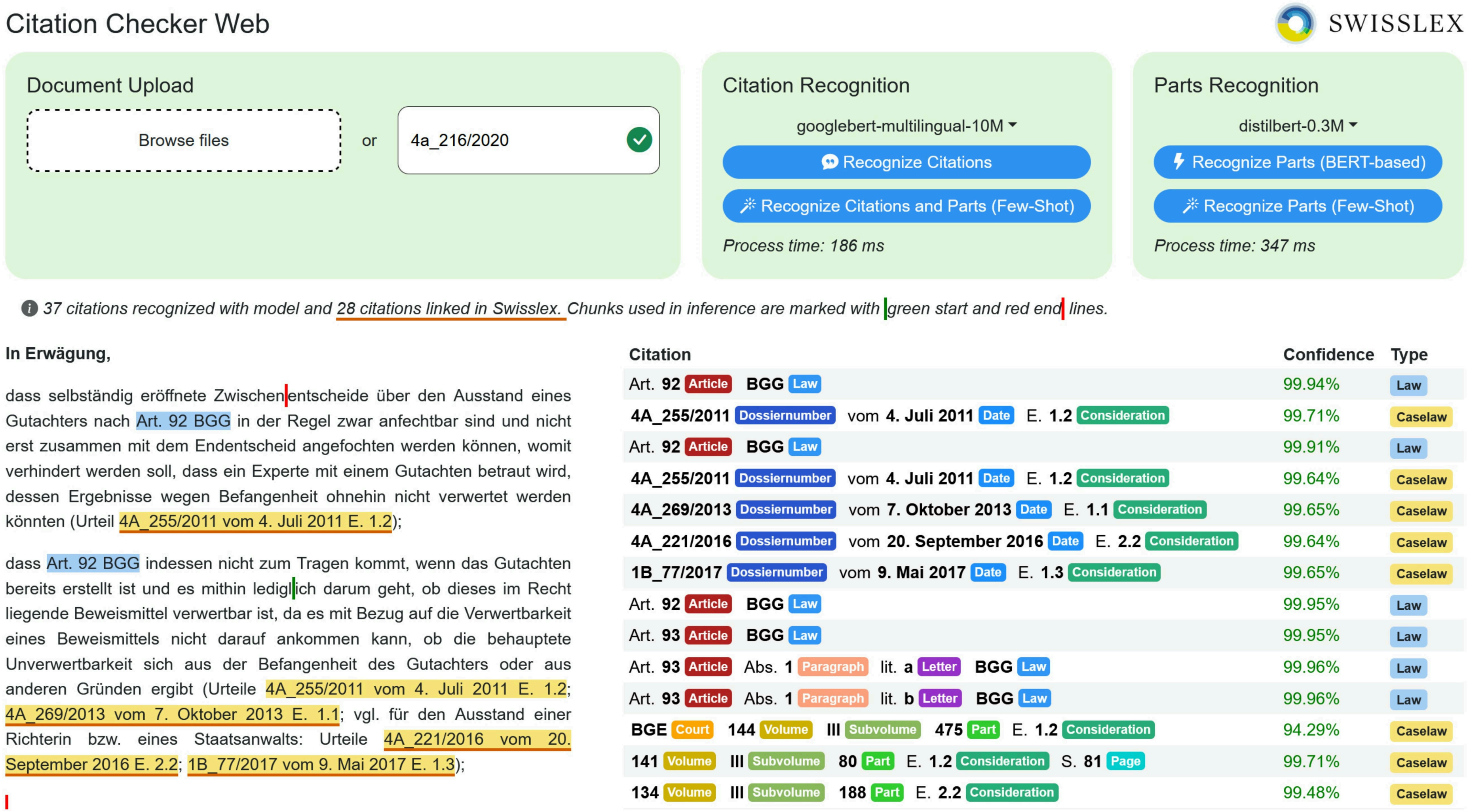



Figure 56: Screenshot of the web application *Citation Checker Web*

## 6.4 Citation Processor

To allow bulk processing of documents, a console application called *Citation Processor* is developed. It is used to recognize citations and their parts in a large number of documents, as it will be done during the nightly re-indexing at Swisslex (chapter 1.3.3).

### 6.4.1 Architecture

Swisslex uses an internal tool, the *Job Scheduler*, to execute tasks like re-indexing documents. The goal of the *Citation Processor* is to later be integrated into the *Job Scheduler* and to allow the recognition of citations and their parts in an automated way. The integration into the *Job Scheduler* is not part of this project (chapter 1.3.4).

The *Citation Processor* contains the `ReferenceRecognitionJob` class, which can be used by the *Job Scheduler* from Swisslex.

To store the found citations and its parts, the database from Swisslex is extended. The new tables are shown in Figure 29 from chapter 5.4. An SQL script to create these new tables is provided for Swisslex, so they can easily update the database when deploying the new version of the software.

Logging is implemented using the log4net library [81]. The log messages are written to the console and to a log file. The log level and other configurations can be set in the `log4net.config` file. A new log file is created with every new program execution. Error handling is implemented so that the program does not crash when an error occurs. Instead, the error is logged and the program continues to process the next document.

### 6.4.2 Process

Figure 57 displays the process of recognizing citations and its parts. The *Job Scheduler* is grayed out in the sequence diagram, as it is not adjusted in this project. The following steps will be executed:

1. The *Job Scheduler* will handle which documents needs to be processed. The IDs of these documents are loaded and batched.
2. For each batch of IDs, the documents are loaded from the database.
3. Each document of the batch is sent to the *Citation Inference API* to recognize the citations.
4. All found citations of the document batch are gathered. This list is then sent in batches to *Citation Inference API* to recognize the parts of the citations.
5. After the parts are recognized, the citations of the current document batch and their parts are stored in the database.
6. These steps are repeated until there are no more documents to process.

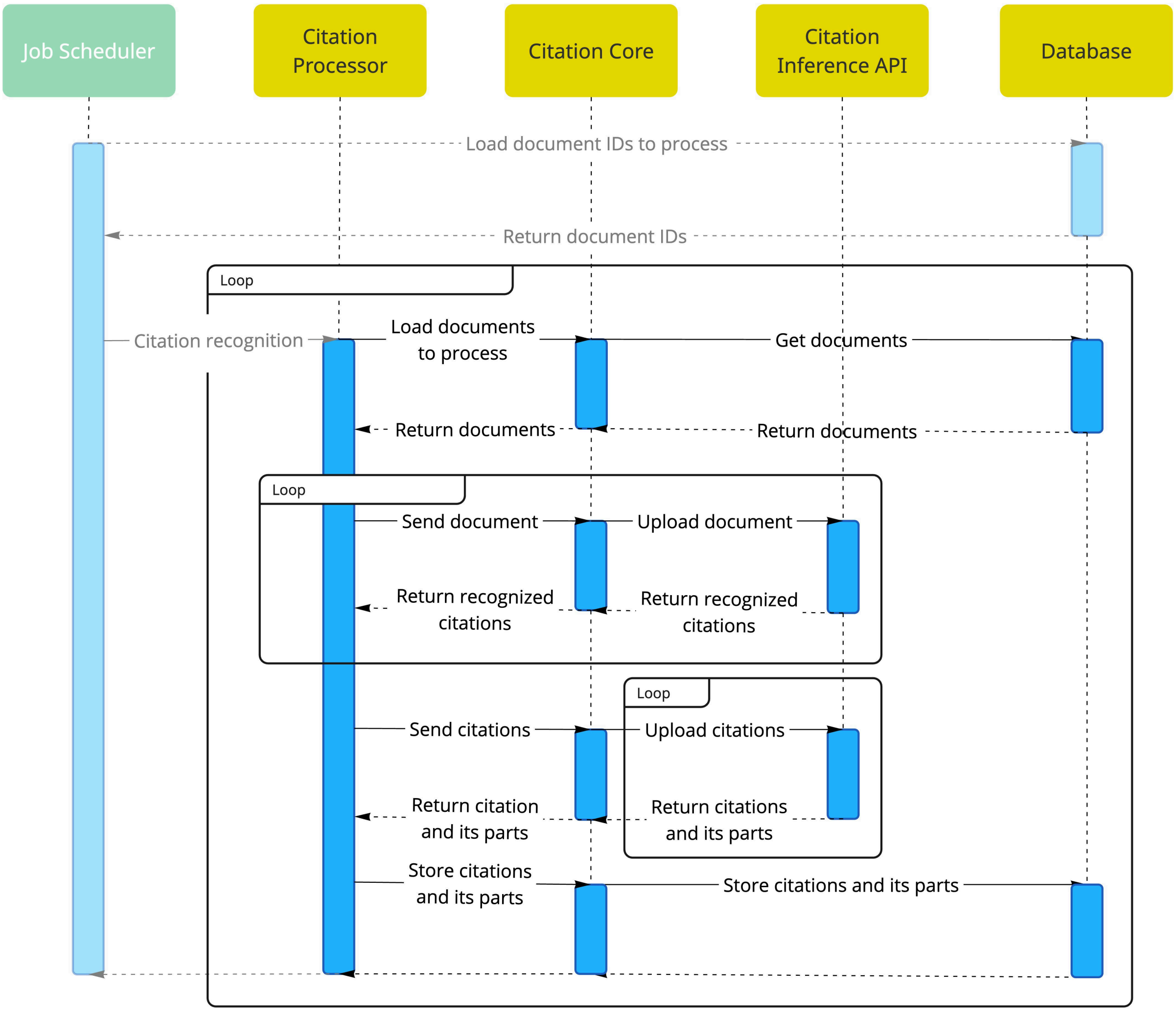



Figure 57: Sequence diagram - Citation Processor

### 6.4.3 Configuration

The *Citation Processor* can be configured with two parameters. They are shown in Code-Snippet 15.

**`DocumentBatchSize`** Defines the number of documents to be processed in a single batch. This configuration helps optimize memory usage. All citations and their parts are stored in the database in a single unit of work.

**`CitationBatchSize`** Configures the number of citations which are sent to the *Citation Inference API* for recognizing their parts in one batch. This reduces the request size.

appsettings.json
```json
"CitationProcessorConfiguration": {
  "DocumentBatchSize": 5,
  "CitationBatchSize": 5
}
```

Code-Snippet 15: Configuration of Citation Processor

## 6.5 Citation Inference API

The *Citation Inference API* is developed as part of this project. It provides functionality for performing inference with fine-tuned models to recognize citations or their components in one or multiple documents.

### 6.5.1 Endpoints

The API is built using the FastAPI framework and is documented using OpenAPI [130]. It provides the following endpoints:

**`POST /v1/predict`** Returns recognized citations or parts for a given document or citation, depending on the specified `type` (`citations`, `parts`) and the model `name` (e.g., `googlebert-multilingual-10M`). An example of the request body is provided in Code-Snippet 16.

```
{
  "text": "... Gutachters nach Art. 92 BGG in der Regel ...",
  "name": "googlebert-multilingual-10M",
  "type": "citations"
}
```

Code-Snippet 16: Citation Inference API - prediction endpoint body example

**`POST /v1/predict_batch`** Similar to the `/v1/predict` endpoint, but designed for a list of documents or citations. An example of the request body is shown in Code-Snippet 17.

```
{
  "texts": [
    "Art. 92 BGG",
    "..."
  ],
  "name": "distilbert-0.3M",
  "type": "parts"
}
```

Code-Snippet 17: Citation Inference API - batch prediction endpoint body example

**`GET /v1/models`** Returns a list of available models for recognizing citations or parts.

The endpoints are additionally shown in Figure 58.

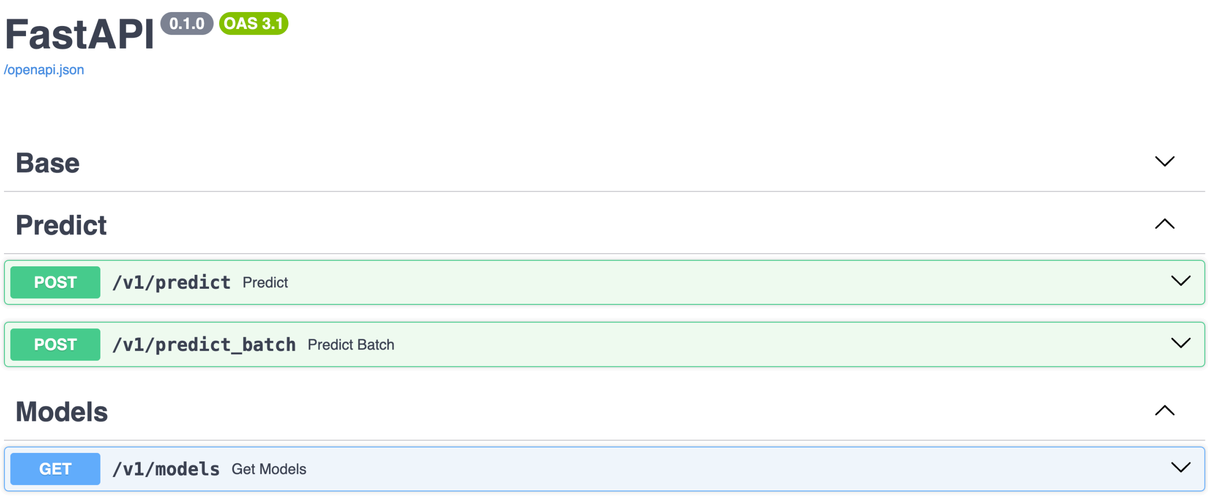



Figure 58: OpenAPI Swagger UI - Citation Inference API

### 6.5.2 Interaction with the API

The sequence diagram in Figure 59 shows the interaction between *Citation Core* and the *Predict Controller*. *Citation Core* initiates the process by sending a request to the `/v1/predict_batch` endpoint of the *Predict Controller* to recognize parts for a batch of citations. The *Predict Controller* responds by loading the fine-tuned model and then invoking it to recognize parts. Finally, the predicted parts are returned to *Citation Core*.

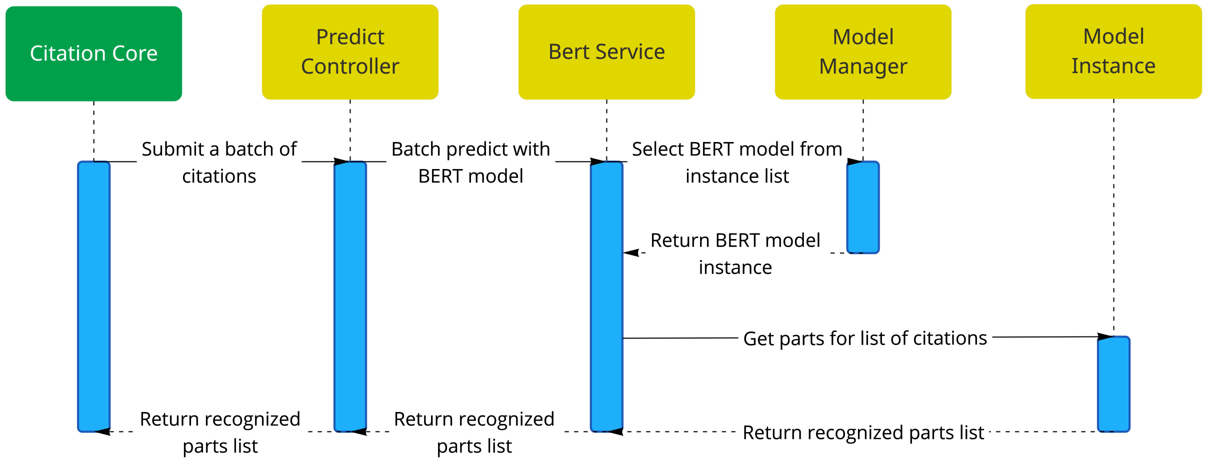



Figure 59: Sequence diagram - Citation Inference API example interaction

### 6.5.3 Inference

The inference process for recognizing citations is performed using the Hugging Face pipeline for NER.

The pipeline is initialized with the fine-tuned model and tokenizer. Due to the limited context size of the models, the training data is split into chunks of 1′000 characters, as described in chapter 6.1.2.4. To ensure that the input during inference is comparable to the training data, a sliding window approach is used. This means that the text is split into overlapping windows, and the model is applied to each window. The results are then aggregated to get the final predictions. This also helps to avoid truncating citations that occur at the beginning or end of a chunk.

The sliding window functionality is not implemented manually, but the corresponding functionality from the Hugging Face Transformers library [131] is used. The `stride` parameter in the `pipeline()` is set to 128, meaning each window overlaps the previous one by 128 tokens. As aggregation strategy, `average` is used for word based models. Microsoft's DeBERTa model and Facebook's XLM-RoBERTa model do not support this, hence `simple` is used for these models. A `model_max_length` of 512 is also specified to define the maximum token length for each window. The Code-Snippet 18 shows how the inference is done for BERT-based models.

```
tokenizer = AutoTokenizer.from_pretrained(model_path, model_max_length=512)

classifier = pipeline(
    "ner",
    model=model_path,
    tokenizer=tokenizer,
    device=get_device(),
    aggregation_strategy="average",
    stride=128,
)
```

Code-Snippet 18: Inference with the Hugging Face NER pipeline

The Figure 60 shows the data flow during citation recognition inference. The text is tokenized and the model predicts the labels for each token. The aggregated output from the classification pipeline is then post-processed and mapped to a new format to represent the recognized citations. The result is returned to the requester in this new format.

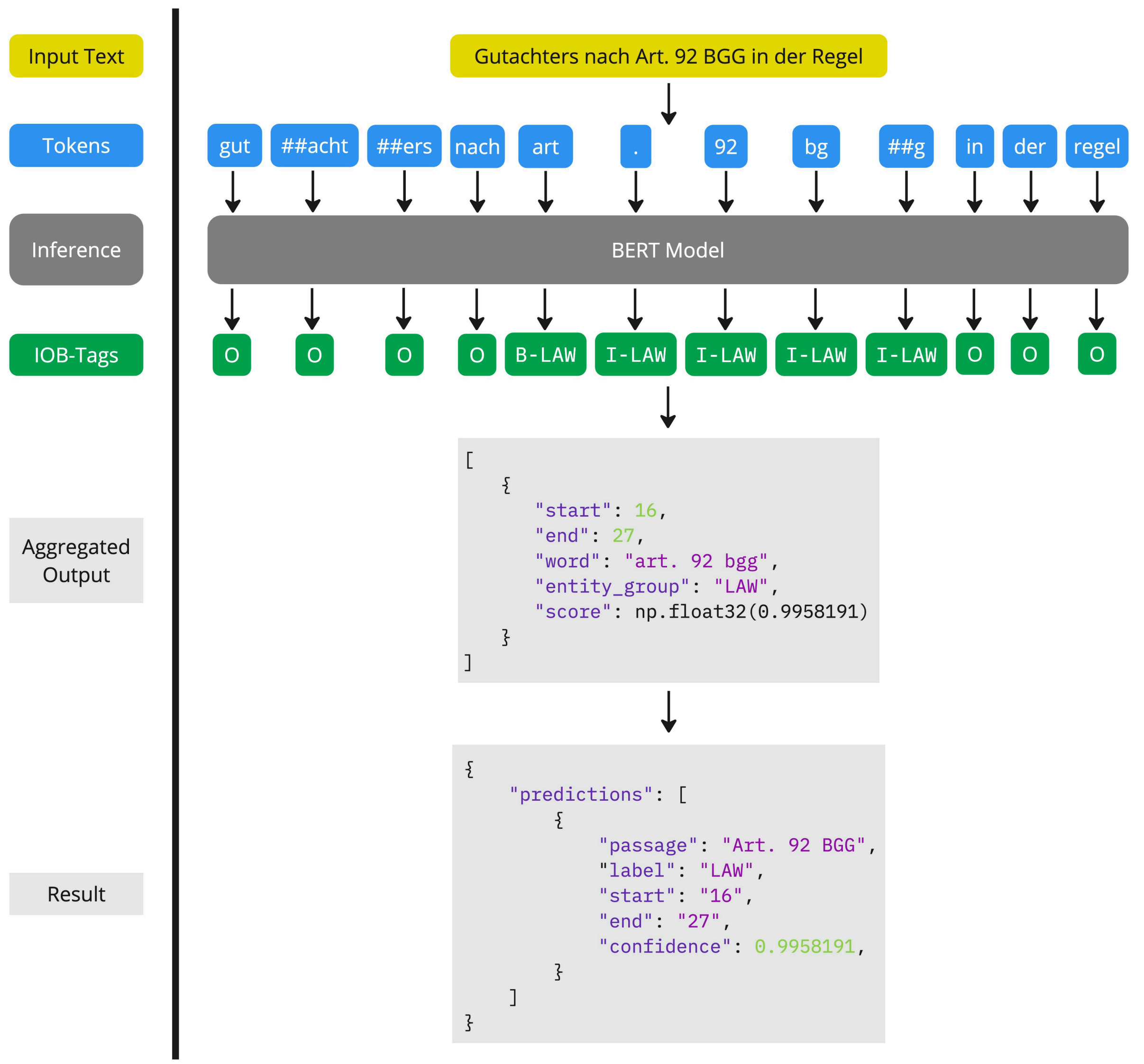



Figure 60: Data flow during citation recognition inference

# 7 Validation

In this chapter, the final solution is validated. First, it is compared to the existing regex-based solution. Afterwards, comparisons against off-the-shelf solutions are made, the testing concept is highlighted, and the requirements are validated.

## 7.1 Comparison with Regex Solution

The currently used regex solution is the benchmark for the new approach to recognize citations. The goal is to achieve similar or better results with the Google BERT multilingual uncased model. The following sections compare the performance of the two solutions on different test sets.

### 7.1.1 Evaluation on Golden Test Set

When evaluated on the golden test set, the regex-based solution achieves an F1 score of 80.55%. The Google BERT base multilingual uncased model, fine-tuned on 10 million samples, scores a slightly higher score of 81.84%. When comparing the recall, the Google BERT model performs approximately 2.2% better. The exact evaluation on the golden test set can be seen in Table 33.

| Model | # of Samples | Precision | Recall | F1 |
|---|---|---|---|---|
| google-bert/bert-base-multilingual-uncased | 10 million | 0.9103 | 0.7433 | 0.8184 |
| current regex solution | - | 0.9117 | 0.7215 | 0.8055 |

Table 33: Evaluation on the golden test set

When further evaluating the current regex solution, it is evident that it also performs better on `CASELAW` and `LAW` citations and worse on `LITERATURE` entities. The Google BERT model shows similar results. Figure 61 shows the F1 score comparison between the regex solution and the fine-tuned Google BERT model. The recall comparison can be seen in Figure 62.

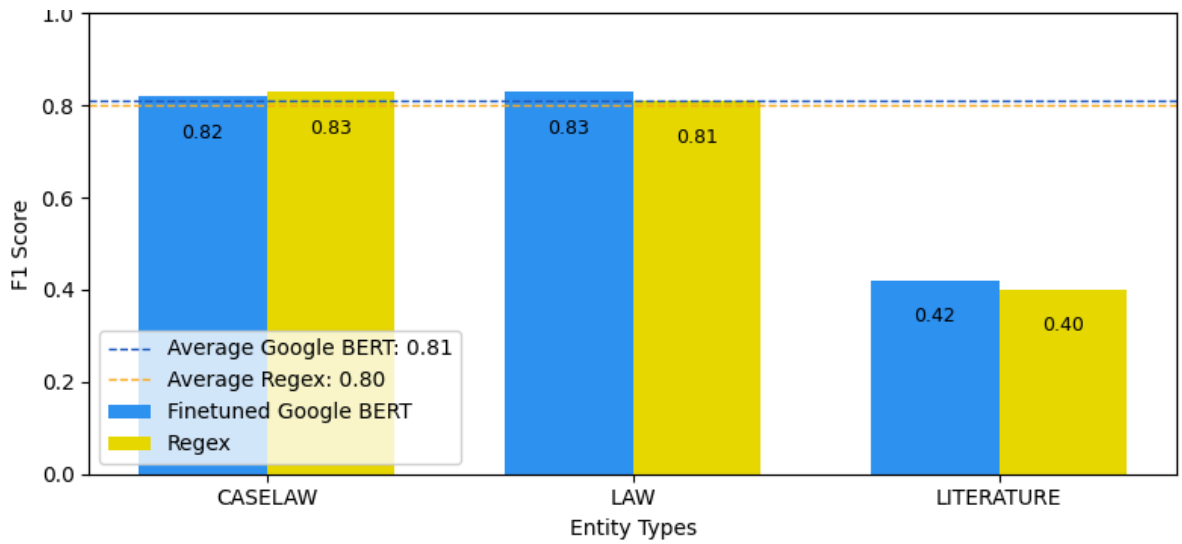



Figure 61: F1 score comparison of the regex solution and the fine-tuned Google BERT model

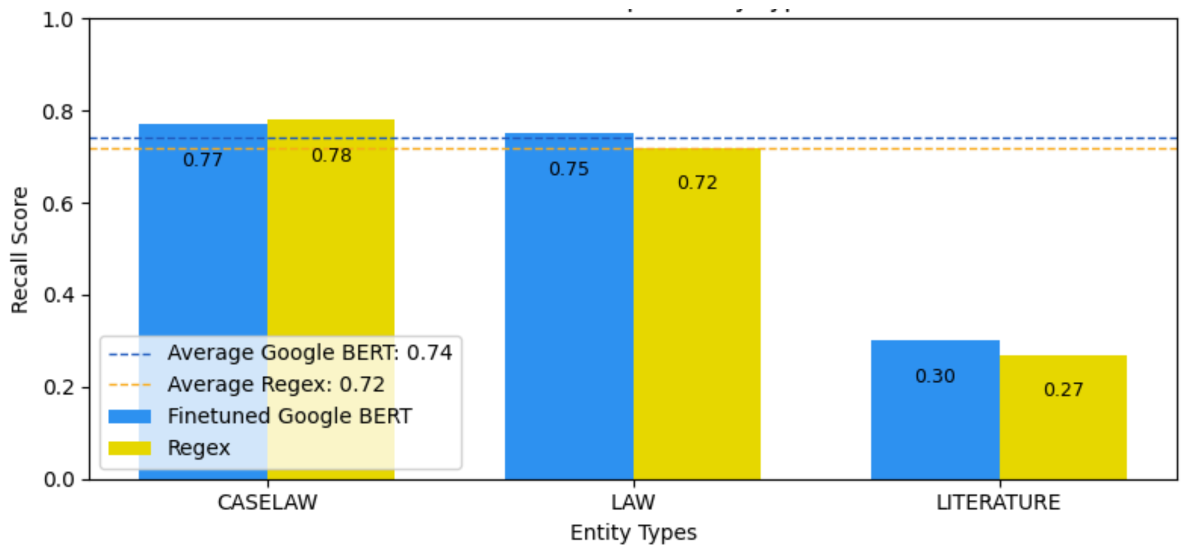



Figure 62: Recall comparison of the regex solution and the fine-tuned Google BERT model

Figure 63 shows the evaluation matrix for the regex solution. It allows for a more detailed insight into the performance. While the precision is high for all entities, the recall is lower, especially for `LITERATURE` entities. For `CASELAW` and `LAW` entities, many partials are recognized.

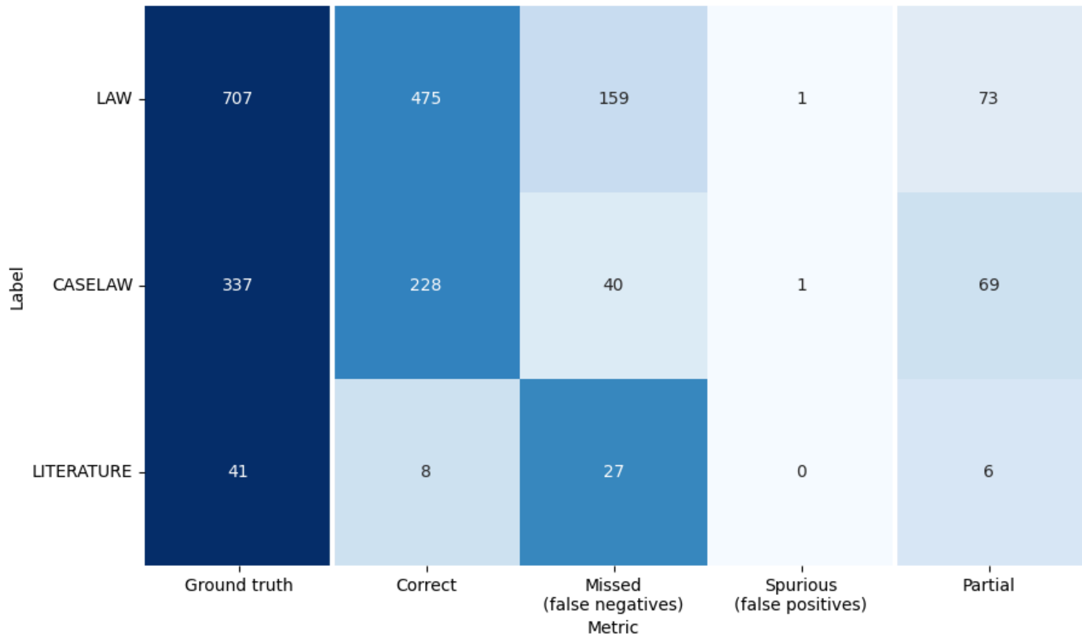



Figure 63: Evaluation matrix for the regex solution

### 7.1.2 Evaluation on Special Cases

Since the previous tests are performed on a random sample of the data, the models and the regex solution are additionally evaluated on a small dataset containing special cases. The special cases are explained in chapter 3.3.4. The results of the evaluation can be seen in Table 34.

It is remarkable that the precision is significantly higher than the recall, for both the Google BERT model and the regex solution.

The Google BERT base multilingual uncased model achieves a slightly higher F1 score than the regex solution. However, the scores show such a small difference that it might be due to random fluctuations.

| Model | # of Samples | Precision | Recall | F1 |
|---|---:|---:|---:|---:|
| google-bert/bert-base-multilingual-uncased | 10 million | 0.9046 | 0.3541 | 0.5089 |
| current regex solution | - | 0.9368 | 0.3485 | 0.5081 |

Table 34: Evaluation on special cases

The evaluated special cases are analyzed in the following sections.

#### 7.1.2.1 Multiple Citations

Manual testing with an example document has shown that the Google BERT model still struggles sometimes with recognizing multiple citations. In the following example, law citations are recognized correctly, but multiple citations of judgments are recognized only partially or as two separate citations. The recognized citations from the model are highlighted in yellow.

- BGE 132 III 18 E. 4.1 S. 20; 132 V 321 E. 6.1 S. 326
- BGE 133 V 9 E. 3.1 S. 11; 131 III 189 E. 2.6 S. 196
- BGE 121 III 60 E. 3d S. 63; 132 III 503 E. 3.3 S. 508/509
- Art. 63 Abs. 2 und Art. 64 OG
- Art. 156 Abs. 1 und Art. 159 Abs. 1 und 2 OG
- Art. 157 und Art. 159 Abs. 6 OG

#### 7.1.2.2 Ambiguity of Abbreviations

Regarding the ambiguity of abbreviations, manual testing has shown that the Google BERT model can clearly distinguish between the two meanings of “CO” vs. “CO 1”.

#### 7.1.2.3 Cantonal Law Misinterpreted as Federal Law

An example document is used to test whether cantonal laws are misinterpreted as federal laws. The results show that the Google BERT model does not misinterpret cantonal laws as federal laws but simply fails to recognize cantonal law citations. An example is shown in Figure 64.

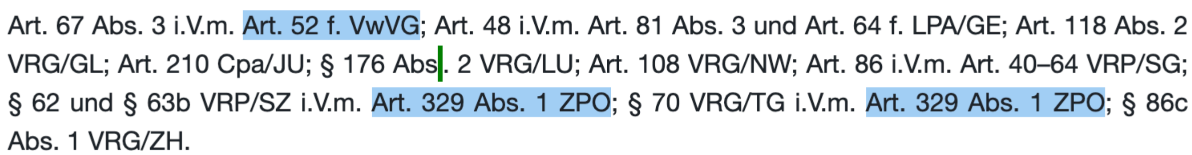

Art. 67 Abs. 3 i.V.m. Art. 52 f. VwVG; Art. 48 i.V.m. Art. 81 Abs. 3 und Art. 64 f. LPA/GE; Art. 118 Abs. 2 VRG/GL; Art. 210 Cpa/JU; § 176 Abs. 2 VRG/LU; Art. 108 VRG/NW; Art. 86 i.V.m. Art. 40–64 VRP/SG; § 62 und § 63b VRP/SZ i.V.m. Art. 329 Abs. 1 ZPO; § 70 VRG/TG i.V.m. Art. 329 Abs. 1 ZPO; § 86c Abs. 1 VRG/ZH.

Figure 64: Example of not recognized cantonal article of laws

#### 7.1.2.4 Misinterpreting Systematic Collections of Law as Laws

An example document is used to test whether systematic collections of law are misinterpreted as individual laws. The results indicate that the Google BERT model does indeed recognize systematic collections as laws. An example is shown in Figure 65.

| Citation | Confidence | Type |
|---|---|---|
| SG 291 | 98.74% | Law |
| SG 790 | 98.96% | Law |
| SG 270 | 98.97% | Law |
| SG 154 | 98.97% | Law |

Figure 65: Example of not recognized cantonal article of laws

#### 7.1.2.5 Deviations from the Standard Citation Format

The fine-tuned Google BERT model can handle citations with a different order of words. For example, the citation reference "ZGB 837 Abs. 1 Ziff. 3" is recognized correctly as a citation, despite the correct citation being "Art. 837 Abs. 1 Ziff. 3 ZGB".

### 7.1.3 Robustness

One limitation of the regex solution is its inability to handle typographical errors. The new citation recognition system is more robust and can handle a certain degree of such errors.

This is manually tested by slightly modifying a citation in a text. The text is uploaded to the web interface. Then, the model output for citation recognition is compared with the expected result.

Figure 66 shows an example of recognized citations with typographical errors. The Google BERT model correctly identifies all citations, even with minor deviations such as missing or additional characters.

Nachteil bewirken können (Art. 93 Abs. 1 lit. a BGG) oder wenn die Gutheissung d
Nachteil bewirken können (Art 93 Abs. 1 lit. a BGG) oder wenn die Gutheissung de
Nachteil bewirken können (Art, 93 Abs. 1 lit. a BGG) oder wenn die Gutheissung d
Nachteil bewirken können (Art. 93 Abstz. 1 lit. a BGG) oder wenn die Gutheissung
Nachteil bewirken können (Art. 93 Abss. 1 lit. a BGG) oder wenn die Gutheissung

Figure 66: Example of recognized citations with typographical errors

### 7.1.4 Conclusion

On the task of recognizing citations, the proposed `google-bert/bert-base-multilingual-uncased` model matches the performance of the regex-based solution. On manually annotated data, it achieves a recall of 74.33%, while the regex-based solution scores 72.15%.

The new solution is more robust and can handle deviations such as typing errors and new citation formats more easily. Wrong order of citation parts are handled better. Ambiguous abbreviations can be distinguished based on the context more effectively.

Special cases are still mostly a challenge for the fine-tuned model.

A great improvement over the existing solution is the more granular labels for citation parts.

## 7.2 Comparison with Off-The-Shelf Models

To benchmark the performance of state-of-the-art off-the-shelf models on the citation recognition task, GPT-4o, GPT-4o-mini, and the Llama 3.3 70B model are evaluated on the golden test set. They are used without any prior fine-tuning.

### 7.2.1 Approach

Since Swisslex posesses a substantial amount of licensed data, it is essential to ensure that only publicly available case law assets are used for this evaluation. To prepare the golden test set, all non-case law assets were filtered out, reducing the dataset from 502 chunks to 245. These filtered chunks are then provided to the models for evaluation, using the same prompt outlined in appendix section F.2.3.1 for consistency across all models.

### 7.2.2 Evaluation on Golden Test Set

The evaluation on the golden test set, as presented in Table 35, shows that the fine-tuned Google BERT model outperforms the off-the-shelf models GPT-4o, GPT-4o-mini, and Llama 3.3 70b in both precision and recall. Notably, the GPT-4o model exhibits impressive few-shot prompting capabilities, particularly in terms of recall. However, the fine-tuned Google BERT model surpasses it, achieving approximately 8% higher recall and significantly higher precision of around 25%.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| google-bert/bert-base-multilingual-uncased (fine-tuned on 10mio) | 0.9060 | 0.7058 | 0.7935 |
| gpt-4o (few-shot) | 0.6523 | 0.6271 | 0.6394 |
| gpt-4o-mini (few-shot) | 0.5404 | 0.5806 | 0.5598 |
| llama 3.3 70b (few-shot) | 0.5465 | 0.6083 | 0.5757 |

Table 35: Evaluation state-of-the-art off-the-shelf models on the golden test set

The evaluation matrix for the best performing off-the-shelf model, GPT-4o, is shown in Figure 67.

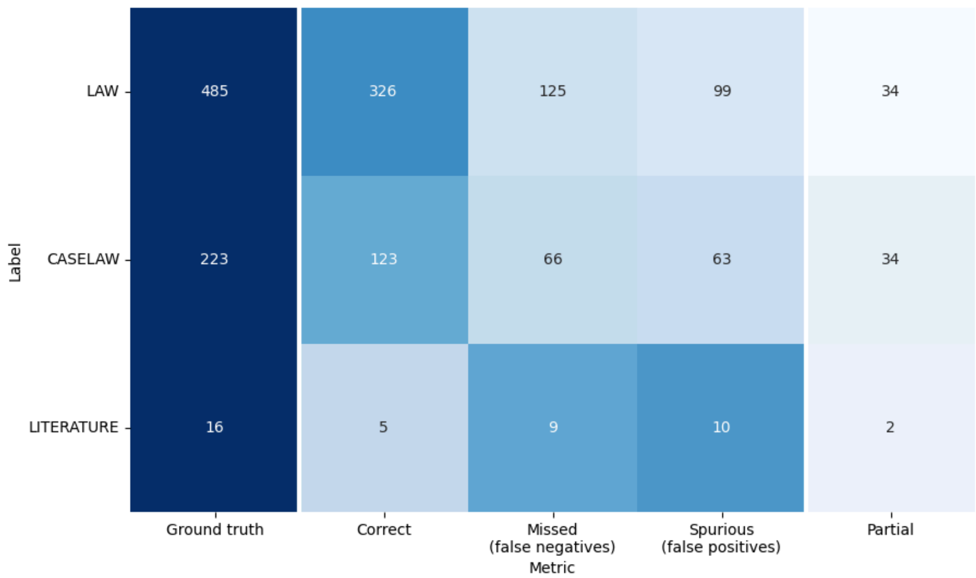



Figure 67: Evaluation matrix for GPT-4o off-the-shelf solution

### 7.2.3 Conclusion

The evaluation results show that the fine-tuned Google BERT model outperforms the tested off-the-shelf models. This demonstrates that fine-tuning is effective and worthwhile.

## 7.3 Evaluation per Language

The citations test set is split by language to evaluate the performance of the models on German, French, Italian, and English. In Table 36, the evaluation metrics per language are presented. The model successfully identifies citations in all languages, with recall values above 90% for each language. This confirms that the goal of robust citation recognition across multiple languages has been successfully achieved.

| Language | Precision | Recall | F1 |
|---|---|---|---|
| German | 0.9595 | 0.9697 | 0.9594 |
| French | 0.9452 | 0.9681 | 0.9565 |
| Italian | 0.9078 | 0.9699 | 0.9378 |
| English | 0.7189 | 0.9280 | 0.8102 |

Table 36: Evaluation per language

## 7.4 Processing Performance

Five experiments are conducted to measure the performance of the citation recognition system. This includes loading the documents from the database, recognizing citations and its parts and storing the results into the database. For each experiment, a different batch of 3′000 documents is processed. The measurement has been made with the *Citation Processor* application (chapter 6.4). The results are shown in Table 37.

NFR-01 (chapter 2.2) specifies a maximum processing time of 7 seconds per document. This target is highlighted in grey in Table 37. The system achieves an average processing time of 0.639 seconds per document, which is well within the required performance parameters.

| Nr | Total Duration | Seconds per Document |
|---|---|---|
| 1 | 32m 57s | 0.659 |
| 2 | 30m 53s | 0.618 |
| 3 | 33m 28s | 0.669 |
| 4 | 32m 35s | 0.652 |
| 5 | 29m 57s | 0.599 |
| **Average** | **31 m 58 s** | **0.639** |
| Target | 6h | 7.000 |

Table 37: Performance measurements

## 7.5 Tests

The following chapters describe the testing concept and the conducted tests together with their results.

### 7.5.1 Testing Concept

This chapter outlines the testing concept for the individual components of the system.

**Models** To test the fine-tuned models, they are evaluated using the `nervaluate` package, which is mentioned in chapter 4.4. On a test set, the model's predictions are compared to the ground truth labels, and the precision, recall, and F1 score are calculated. The evaluation results are then used to determine the model's performance and identify areas for improvement.

**Post-Processing Steps in Inference** The post-processing steps performed during inference are tested using unit tests, since they contain business logic that can be tested in isolation. The steps are highlighted in chapter 6.1.11. The unit tests and their results are documented in chapter 7.5.2.

**GUI and C# Core Library** Since the web interface and the core library barely contain business logic and are mainly used as communication interfaces, they are not tested with unit tests. Instead, manual end-to-end tests are performed to ensure that the GUI functions correctly and that the core library processes the data as expected. They are described in chapter 7.5.3.

**Integration Tests** Integration tests are not required for this project, as the components do not need to be integrated into a larger system.

### 7.5.2 Unit Tests

For the post processing steps (chapter 6.1.11), several unit tests are implemented to ensure the correct functionality. As testing framework, `unittest` [132] is used. The Table 38 shows the implemented tests and their expected behaviour and results. All tests passed successfully.

| Title | Expected Behaviour | Result |
|---|---|---|
| test_deduplicate_entities | Entities with the same start and end tag and label get deduplicated. The one with the higher confidence is kept. | **Passed** |
| test_merge_overlapping_entities | Overlapping entities (one from index 2 to 9 and one from index 4 to 10) with the same label are merged. The confidence is averaged, the maximum length is taken. | **Passed** |
| test_merge_overlapping_entities_same_start | Overlapping entities with the same start index but a different end index and a different label are merged. The longer one (second one) is kept. | **Passed** |
| test_merge_overlapping_entities_same_end | Overlapping entities with the same end index but a different start index and a different label are merged. The longer one (first one) is kept. | **Passed** |
| test_merge_overlapping_entities_fully_enclosed | An entity that is fully enclosed within another one is not kept. | **Passed** |
| test_merge_overlapping_entities_no_overlap | When entities do not have an overlap, they are not discarded during post-processing. | **Passed** |
| test_get_positions_of_passages | Given a text and passages, the correct start and end indices of these passages are found within the text. | **Passed** |
| test_get_positions_of_passages_remove_hallucinations | Given a text and passages, the correct start and end indices of these passages are found within the text. If hallucinations are present, they are kept and the start and end indices 0 are assigned. | **Passed** |
| test_get_positions_of_passages_keep_hallucinations | Given a text and passages, the correct start and end indices of these passages are found within the text. If hallucinations are present, they are removed. | **Passed** |
| test_fake_confidence | If there is no confidence present in the entity, it is added with a value of 0. | **Passed** |
| test_extend_passages_start | Found passages that start in the middle of a word in the text are extended up to the next special character or whitespace in the left direction. | **Passed** |
| test_extend_passages_end | Found passages that end in the middle of a word in the text are extended up to the next special character or whitespace in the right direction. | **Passed** |

Table 38: Unit tests

### 7.5.3 End to End Tests

The conducted end to end tests are listed in Table 39.

They are tested using the following setup:
- Browser: Firefox, Version 133.0.3 (64-Bit)
- Hardware: Laptop Dell Precision 3551
- Operating System: Microsoft Windows 10 Pro
- Display Resolution: 3840 x 2160 pixels
- Connected with the Swisslex VPN
- Open the web interface under https://citation-checker.swisslex.ch/

| Nr. | Title | Steps | Expected Behaviour | Result |
|---|---|---|---|---|
| 1 | Upload Document | The button "Upload document" is pressed. A text document with the name `test-file.txt`, the mime type `text/plain` and the content `Gemäss BGE 123 V 3 ist das falsch.` is selected. | The name of the file is displayed inside the upload area. The text of the document is displayed. | **Passed** |
| 2 | Search for Dossier Number | Enter the dossier number `4A_216/2024` inside the `dossier number` input field. Press the magnifying glass. | The magnifying glass switches to a green checkmark to indicate that the document was found. The text of the document is displayed. | **Passed** |
| 3 | Recognize Citations | After executing test case 2, the button "Recognize Citations" is pressed. A model is selected from the dropdown. | The citation `BGE 123 V 3` gets highlighted in the text. On the right side, a list with this citation and the label `CASELAW` is visible. | **Passed** |
| 4 | Recognize Parts | After executing test case 3, the button "Recognize Parts" is pressed. | The citation in the list on the right side has its parts labeled. | **Passed** |

Table 39: End to end tests

## 7.6 Validation of Requirements

In the following chapters, the functional and non-functional requirements are validated.

### 7.6.1 Validation of Functional Requirements

In Table 40, the validation of the functional requirements is presented.

| Use Case | Status | Rationale |
|---|---|---|
| Recognize Citations | **Fulfilled** | The comparison of the proposed vs. the regex-based solution in chapter 7.1 showed that the citation recognition works and the new system matches the performance of the regex-based solution. |
| Train Model | **Fulfilled** | In chapter 6.1.4, the training iteration is described, which shows that the training of models is possible. |
| Evaluate Model | **Fulfilled** | In chapter 6.1.4, the training iteration is described, which shows that the evaluation of models is possible. |
| Create Datasets | **Fulfilled** | To create datasets, a program called *Data Extractor* was developed, which extracts the data and creates the needed datasets. This process is described in chapter 6.1.2. |
| Test Citation Recognition | **Fulfilled** | A web application was developed to test the citation recognition system. The application is described in chapter 6.3. |
| Serve Models | **Fulfilled** | The *Citation Inference API* was developed to serve the models. The API is described in chapter 6.5. |
| Store Recognized Citations | **Fulfilled** | New tables were added to the Swisslex database to store the recognized citations. The database changes are described in chapter 5.4. |
| Run Daily Recognition | **Fulfilled** | A new job was created which executes the citation recognition. The job is encapsulated in the console application *Citation Processor*, which is described in chapter 6.4 and can be used by Swisslex's *Job Scheduler*. |

Table 40: Functional requirements validation

### 7.6.2 Validation of NFRs

The non-functional requirements are described in chapter 2.2. The following tables describe the final status of the NFRs.

| **ID** | **NFR-01** Performance |
|---|---|
| **Requirement** | A single legal document (e.g. judgment) should be processed within a reasonable time frame. |
| **Level** | Must have |
| **Status** | **Fulfilled** |
| **Rationale** | The average processing time is **0.639 seconds per document** (chapter 7.4), which is within the specified time frame of 7 seconds per document. |

Table 41: Validation NFR-01 Performance

| | |
|---|---|
| **ID** | **NFR-02** Scalability |
| **Requirement** | The performance of citation recognition should be scalable and still accurate with increasing document throughput. |
| **Level** | Must have |
| **Status** | **Fulfilled** |
| **Rationale** | The citation recognition system processed 3′000 documents in an average of 31 minutes and 58 seconds in 5 experiments (chapter 7.4). It achieves an F1 test score of 95.7% when recognizing citations and an F1 score of 81.8% on the golden test set. |

Table 42: Validation NFR-02 Scalability

| | |
|---|---|
| **ID** | **NFR-03** Compatibility |
| **Requirement** | The solution must be deployable within the Swisslex infrastructure. |
| **Level** | Must have |
| **Status** | **Fulfilled** |
| **Rationale** | The citation recognition job inherits the base job class from Swisslex. This ensures compatibility with the existing job scheduler software. The Citation Inference API provides access to the fine-tuned model for citation recognition. |

Table 43: Validation NFR-03 Compatibility

| | |
|---|---|
| **ID** | **NFR-04** Reliability |
| **Requirement** | The solution should be robust against variations, new patterns, and typos. |
| **Level** | Must have |
| **Status** | **Fulfilled** |
| **Rationale** | By using Transformer-based models and replacing the regex-based solution, the citation recogntion system is more robust. |

Table 44: Validation NFR-04 Reliability

| | |
|---|---|
| **ID** | **NFR-05** Security |
| **Requirement** | The solution should ensure data privacy and security. |
| **Level** | Must have |
| **Status** | **Fulfilled** |
| **Rationale** | The provided solution uses inhouse trained models which are deployed inside the Swisslex environment. No data leaves this environment during citation recognition. For evaluating off-the-shelf models like CHatGPT, only judgments were used since they are publicly available. No licensed data was shared with third parties. |

Table 45: Validation NFR-05 Security

| | |
|---|---|
| **ID** | **NFR-06** Usability |
| **Requirement** | Easy to use testing software for citation recognition. |
| **Level** | Should have |
| **Status** | **Fulfilled** |
| **Rationale** | A web interface enables uploading a document or loading a document based on its dossiernumber from the database. The user is lead through the process by a 3-step wizard-like interface. Found citations get highlighted in the document and are displayed in a separate table. |

Table 46: Validation NFR-06 Usability

# 8 Discussion of Results and Outlook

This chapter discusses the results of the project and compares them with the objectives set in the introduction. Also, further improvements and recommendations for future work are outlined.

## 8.1 Results

In this bachelor thesis, a system for the recognition of citations and their parts in legal documents was developed. The goal was to replace the existing regex-based solution with a modern transformer style NLP-based solution. The currently recognized citations extracted by the regex solution were used as training and test data.

To recognize citations, different models were fine-tuned and evaluated on the three classes `CASELAW`, `LAW` and `LITERATURE`. The models evaluated ranged from generative models such as Llama to BERT-based models of different sizes and architectures. They were trained in a supervised manner and evaluated on metrics such as precision, recall, F1 score and inference time. In addition, an ensemble of label-specialist models was evaluated but not pursued further due to a high number of false positives.

In the end, the best performing model was `google-bert/bert-base-multilingual-uncased`. The model was trained on 10 million samples and achieved an F1 test score of 95.74% across all labels. Broken down into the three labels, the F1 test score is 95% for `CASELAW`, 96% for `LAW` and 85% for `LITERATURE`.

The new system matches the performance of the regex-based solution. When compared on the manually annotated golden test set, the new model slightly outperforms the existing regex-based solution (F1 score of 81.84% vs. 80.55%, recall of 74.33% vs. 72.15%). This demonstrates the potential of the new system for improved citation recognition in legal documents.

For the specific parts of the citations, there was not enough training data to fine-tune a model. Therefore, few-shot prompting with Llama 3.1 8B model was set up. Although it gave good results, the model could not be used in the final solution due to the slow inference time.

Instead, the few-shot prompts were used to generate training data for parts recognition. For the knowledge distillation process, the 8B variant was replaced with Llama 3.3 70B, as this approach yielded superior results. In the end, about 380′000 chunks of training and test data were generated. This data was used to train a separate DistilBERT model to recognize parts with the labels `COURT`, `PUBLISHER`, `VOLUME`, `SUBVOLUME`, `PART`, `CONSIDERATION`, `PAGE`, `DATE`, `DOSSIERNUMBER`, `YEAR`, `AUTHOR`, `TITLE`, `DECREEFORM`, `REFERENCE`, `ARTICLE`, `PARAGRAPH`, `SENTENCE`, `HALFSENTENCE`, `NUMBER`, `LETTER` and `LAW`.

The model achieved an F1 test score of 98.9% for recognizing citation parts. On the manually annotated test set, an F1 score of 93.60% and recall of 93.37% was achieved.

The citation recognition solution can process 3′000 documents in approximately 32 minutes. This process includes identifying citations and their individual parts. The system achieves an average processing time of 0.659 seconds per document. With a specified time frame of 7 seconds per document, the solution operates efficiently and well within the required time constraints.

The citation recognition pipeline can be accessed via a web application or a console application. The web application allows the user to upload or search for a document in the Swisslex database, which is then processed by the pipeline. The result is visualized in the application, showing the recognized citations and parts. This allows to easily verify and evaluate the system.

The console application can be used to process several documents from the database, for example during the nightly re-indexing of the Swisslex database. The recognized citations and parts are stored in the database in a structured way and can be used for further processing.

This solution meets the requirements of the task and provides a good trade-off between performance and resource consumption.

On the task of recognizing citations, the proposed `google-bert/bert-base-multilingual-uncased` model matches the performance of the regex-based solution while offering several key advantages. Unlike regex, which relies on rigid patterns, the model is more robust and adaptable. It can effectively handle deviations such as typographical errors or non-standard citation formats. It can also differentiate ambiguous abbreviations based on context. Furthermore, citation parts recognition has been significantly enhanced with the introduction of more granular labels. 21 different citation parts can be recognized, allowing a more detailed understanding of citations. The flexibility and scalability of the machine learning approach ensure that the system can be retrained efficiently to incorporate additional citation types or adapt to evolving legal citation standards. Together, these advancements make the proposed solution not only more precise but also more future-proof compared to traditional rule-based methods.

Challenges remain for certain special cases. To optimize the results further, there are some improvements that can be made. These possibilities will be discussed in the next chapters.

## 8.2 Limitations

The proposed solution has several limitations that need to be considered when interpreting the results.

### 8.2.1 Infrastructure Setup

The models are hosted on the DGX server from Swisslex. The DGX is part of a newly set up Kubernetes cluster and is not yet fully operational. This has the following implications for the inference of models:

- **Silent request failures**: When querying the *Citation Inference API*, requests frequently fail silently. They never reach the API, and no exception is thrown. The failure is probably due to misconfigurations in the Kubernetes cluster, which are not in the control of the client. To handle this, there is a client side timeout configured, after which an exception is displayed in the UI.
- **Gateway timeout**: Certain requests to the *Citation Inference API* take a long time to process, especially when querying the Llama models. After 60 seconds, a gateway timeout is thrown. This is a server side timeout and cannot be controlled by the client.
- **JupyterLab Deployment**: The API is only deployed on JupyterLab, which is not the final deployment target.

The infrastructure setup is out of the scope of this thesis. There is no possibility to debug or improve the setup, as the infrastructure is provided by Swisslex and is still in development. These limitations have to be kept in mind when using the web application to test the models.

### 8.2.2 Context Size

The models are fine-tuned on chunks of approximately 1000 characters. They include the entire context of each chunk to make predictions. This means that when queried with a single citation, without any context, the model might not recognize the citation.

This limitation becomes relevant in edge cases when processing documents. For example, if a document is divided into multiple chunks and the last chunk is shorter than 1000 characters, the model may struggle to recognize citations in that final chunk due to the reduced amount of contextual information.

This limitation should also be kept in mind when manually uploading texts in the web application, as shorter or incomplete chunks may impact the model's accuracy.

### 8.2.3 Test Scores

The test data is based on the regex solution. If the model generalizes better and recognizes more citations not covered by the regex solution, it could get punished for this in the evaluation. A high F1 score based on the test set does not mean that the model recognizes all citations perfectly. First and foremost, it shows that the model recognizes citations just as good as the regex solution.

### 8.2.4 HTML Tags in Citations

The web application shows the user the formatted version of the document text, which is not equivalent to the text that the model sees. If a document is in HTML format, the model sees the HTML tags, which the user does not see. This could make it hard to understand why the model does not recognize a citation correctly. Since some of the Swisslex documents contain HTML tags within the citations, such as `<i>` tags, the model might struggle to recognize the citations correctly.

E.g. the citation "`BGE 123 IV 1`" could be written as "`BGE <i>123</i> IV 1`" in the document. The model could struggle recognizing this as a citation because of the `<i>` tags. It would be an option to not display the formatted version of the document text to the user, but the raw text. However, due to usability reasons, this was not implemented.

## 8.3 Possible Further Improvements

Since the time of this bachelor thesis is limited, there are still many possibilities for further evaluation and improvements. The following recommendations are based on the results of the experiments and the insights gained during the work on this project.

### 8.3.1 Improve Training Data Quality

The most significant improvements to the solution could be achieved by enhancing the quality of the training data. The data is based on the current regex solution and is not manually annotated. Since the regex solution does not catch all citations, the training data is not perfect. This could lead to the model learning wrong patterns and making incorrect predictions.

An LLM could be used to clean up the training data and increase the diversity of the samples. Through few-shot prompting, the LLM could be queried to find incorrect or missing citations in the training data. This could improve performance when recognizing citations if a new model is trained with the cleaned data.

When the citation recognition is improved and more specific components like halfsentences, references etc. are recognized, new training data can be generated for parts. The model specialized in recognizing parts can be re-trained on this new dataset.

### 8.3.2 Balance Training Data

The data exploration in chapter 3.3 shows that the data for recognizing citations is not perfectly balanced. For example, documents with the labels `CASELAW` and `LAW` get cited more often than `LITERATURE` documents. This could lead to the model being biased towards the labels which get cited more often. The model could learn to predict the labels with more samples better than the labels with fewer samples. This specific problem is covered in chapter 6.1.10, where the performance of the labels is described. This might be a general issue, e.g. for short and long citations, language specific citations, and so on. Balancing the dataset should also be considered for recognizing parts of a citation.

Based on the structure of the dataset it is difficult to balance it (chapter 6.1.1). The dataset uses a text chunk as starting point. The sample contains the citations which occur in the text chunk. When trying to get more examples for the label `LITERATURE`, a lot of the samples containing `LITERATURE` citations would also contain `CASELAW` and `LAW` citations. Since the results are already satisfying and the balancing takes a lot of time, this is not a priority for this bachelor thesis. However, with more time at hand, this could be a valuable improvement. To improve balance, chunks with `LITERATURE` citations but few or no `CASELAW` and `LAW` citations could be filtered out. Another alternative would be to use an LLM to create synthetic samples.

### 8.3.3 Include Linking Words for Parts of a Citation

For the recognition of parts in citations, labels are defined which represent the different values of the parts (chapter 6.2.1). For example, the label *article* represents the value "Art. 1" and the label *page* represents the value "Seite 12".

This recognition of the parts could be improved. For example, if the text contains "Art. 1 bis 5" it recognizes "Art. 1" and "Art. 5" but not the linking word in between. The contextual information of the word "bis" is not represented by the recognized parts. The same applies to "Seite 12 ff" in which "Seite 12" is recognized but not the meaning of "ff" (=fortfolgend).

Based on the recognized citations and its parts, Swisslex will implement the linking of the citation with its cited document after this thesis. If it turns out that it is not good enough with the provided labels, it may be worth trying to include more labels which represent the linking words or find another solution in which these could be represented. If such a solution is found, it will be easy to update the already recognized citations, since the whole citation passage is stored in the databse, including the linking words.

### 8.3.4 Improve Golden Datasets

The golden datasets are used to evaluate the "real" performance of a fine-tuned model and compare it to the performance of the regex solution (chapter 7.1) or to the few-shot prompting approach (chapter 6.2.2.5). The golden datasets are created by manually annotating the citations and their parts in the text, as described in chapter 6.1.2.7. The annotation is done by the authors of this work and is best effort. This means that the annotations may not be perfect and may contain errors, as the authors are not legal experts. It is therefore recommended that the golden datasets are reviewed and expanded by legal experts. This would ensure that the annotations are correct and a larger dataset could be created.

### 8.3.5 Experts per Label

As described in chapter 6.1.10, the performance of the ensemble of label-specialist models was not as good as expected. This was mainly due to the high number of false positives, which could be explained by the fact that the training sets only contained positive samples. If the datasets could be created in a more balanced way, still including many positive samples but also some negative ones where the label is not present, the performance of the ensemble of label-specialist models could be improved. However, it still has to be evaluated if it would perform better than a single model.

### 8.3.6 More Specific Labels for Citations

The labels for citations in legal documents could be further divided into more specific labels. This idea came up because of the different citation formats for judgments. For example the Federal Supreme Court of Switzerland uses the format "BGE 123 IV 1" for their published judgments and the administrative court of the canton of Zurich uses the format "VB.2024.00535".

This thesis uses the labels `CASELAW`, `LAW` and `LITERATURE`. But because of those different formats in citations it may be beneficial to further divide the labels. For example, the label `CASELAW` which represents judgments could be divided into european, national and regional judgments.

### 8.3.7 Larger Context Size

The chunk size of 1000 characters is chosen because of the context size limitations of BERT-based models. However, Llama models can handle larger context sizes. It could be evaluated whether a larger context size would improve the performance of the model. A quick test showed that increasing the context size during inference did not improve performance. This is explained by the fact that the model was fine-tuned for chunks of 1000 characters. The model would have to be fine-tuned again with a larger context size to benefit from this.

### 8.3.8 Adjust Tokenizer

The tokenizer could be improved by including abbreviations of commonly referenced laws, such as *OR* or *ZGB*, as separate tokens. These abbreviations are unlikely to be part of the standard vocabulary of the tokenizer, but are an integral part of Swiss legal texts. By explicitly adding these terms, the tokenizer would reliably identify the boundaries of legal references. For example, in a citation such as *Art. 123 ZGB*, the tokenizer would correctly isolate *ZGB* as a specific legal code, instead of breaking it down into unrelated subwords or treating it as part of the surrounding context.

This phenomenon is not explicitly observed during the project. However, due to the large number of abbreviations, it is not feasible to manually check the tokenization of each abbreviation. Nevertheless, false negatives could be evaluated and checked to see if they are caused by the tokenizer.

### 8.3.9 ModernBERT

In the end phase of this bachelor thesis on December 19 2024, a new BERT based model was released called ModernBERT from B. Warner *et al.* [133]. Prof. Dr. Mitra Purandare, the advisor of this thesis, mentioned that this model could be interesting for this thesis. A short evaluation has shown that on 100k training samples (with a 80/20 validation split) and 30k test samples only an **F1 score of 45.5%** could be reached (detailed information in appendix section F.2.4). This is not enough for the use in this thesis and because the time is limited, further evaluation is not possible. Still, it could be worth to evaluate this model again in the future, especially since the context size of ModernBERT is extended to 8K tokens.

### 8.3.10 Directly Recognize Parts

The proposed solution splits the task into two steps, recognizing citations and then afterwards their parts. Inference time could be reduced when merging these steps and directly recognizing the parts of a citation. The challenge with this approach lies in determining which parts belong to the same citation, especially when the parts are not directly adjacent to each other or are part of a multiple citation. An option for this could be nested NER from GLiNER [38].

### 8.3.11 Few-Shot Prompting for Parts

In the future, as LLMs become faster and cheaper to run, or smaller models become better, few-shot prompting with an LLM could replace the fine-tuned BERT model for recognizing parts in citations. This could improve generalization of unseen data.

## 8.4 Acknolwedgements

We would like to thank our advisors, Prof. Dr. Mitra Purandare and Benjamin Plattner, for their support and valuable feedback throughout the project. We are grateful for the time and effort they invested in reviewing our work. In particular, we would like to thank Prof. Dr. Mitra Purandare for accepting to supervise our self-submitted project.

We would also like to thank our practice partner Swisslex - Schweizerische Juristische Datenbank AG for trusting in us to work on this project and for providing us with the necessary resources and information. A special thanks to Stefano Branco and Christian Niklaus for their guidance and for answering our questions.

We also want to thank Marcel Amsler from Sanitas and Urs Baumann from OST for their work as co-examiner and proofreader, respectively.

Additionally, many thanks to our friends and family for their support during the project and for proofreading this thesis.

# A Glossary

**BiLSTM-CRF** Combination of Bidirectional LSTMs with CRFs
**BERT** Bidirectional Encoder Representations from Transformers
**Case law** Law which originates from court decisions. This term is used in Swisslex and in this thesis interchangeably with judgment.
**Chunk** A part of a document which is processed by the model
**Citation Checker Web** Web application to test the citation recognition
**Citation Inference API** API to perform inference with the models for citation recognition
**Citation label** Label to classify the citations, either `LAW`, `CASELAW` or `LITERATURE` (chapter 6.1.1)
**Citation Processor** Console application which allows bulk processing of documents
**CRF** Conditional Random Field
**Data Extractor** Helper tool to extract data from the Swisslex database
**DGX** *"The Nvidia DGX represents a series of servers and workstations designed by Nvidia, primarily geared towards enhancing deep learning applications through the use of general-purpose computing on graphics processing units (GPGPU)."* [134]
**F1** Score to evaluate model performance, calculated as the harmonic mean of precision and recall
**FN** False Negative
**FP** False Positive
**Golden test set** Test set which is manually annotated and is used to evaluate the model
**HMM** Hidden Markov Model
**IOB2** Inside-Outside-Beginning (IOB) tagging scheme
**KNN** K-Nearest Neighbors
**L2** Regularization term which penalizes large weights
**LLM** Large Language Model. In this thesis, the term refers to decoder-only models with over 1 billion parameters, excluding BERT-based models and similar encoder architectures.
**LR** Learning Rate
**LSTM** Long Short-Term Memory
**NER** Named Entity Recognition
**NLP** Natural Language Processing
**Part Type** Type of a part inside a citation, described in detail in chapter 6.2.1
**POS** Part-of-Speech
**Production Tool** Internal tool used by Swisslex to process legal documents
**Regex solution** Currently used solution for citation recognition in Swisslex, which is based on regular expression matching
**RNN** Recurrent Neural Network
**RUP** Rational Unified Process
**Special case test set** Test set which is created and manually annotated to evaluate the performance of the model on edge cases
**SLX** Internal abbreviation for Swisslex
**Swisslex** Online platform for searching legal documents, also the name of the company which provides the platform
**SVM** Support Vector Machine
**TN** True Negative
**TP** True Positive

# B List of Tables

# C List of Illustrations

# D List of Code

# E Bibliograpy

[1] "Willkommen bei Swisslex," *Swisslex - Schweizerische Juristische Datenbank AG.* Accessed: Sep. 23, 2024. [Online]. Available: https://www.swisslex.ch/de/product/overview

[2] "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)*, Association for Computational Linguistics, pp. 54–59. doi: 10.18653/v1/N19-4010.

[3] "Modulbeschreibung: Bachelor-Arbeit Informatik," Sep. 20, 2024, *OST – Ostschweizer Fachhochschule.* Accessed: Jul. 02, 2024. [Online]. Available: https://studien.ost.ch/allModules/41626_M_BAI21.html

[4] "Swisslex." Accessed: Sep. 23, 2024. [Online]. Available: https://www.swisslex.ch/

[5] "Zitierregeln," Jan. 2021, *Schweizerisches Bundesgericht.* Accessed: Sep. 30, 2024. [Online]. Available: https://www.bger.ch/files/live/sites/bger/files/pdf/Reglemente/01_Zitierregeln_2021_d.pdf

[6] "Entscheidungen des Schweizerischen Bundesgerichts," Feb. 19, 2023, *Wikipedia.* Accessed: Sep. 30, 2024. [Online]. Available: https://de.wikipedia.org/wiki/Entscheidungen_des_Schweizerischen_Bundesgerichts

[7] "Merkblatt für die Anfertigung einer Seminararbeit im Arbeitsrecht," Dec. 07, 2020, *Universität Zürich.* Accessed: Dec. 02, 2024. [Online]. Available: https://www.ius.uzh.ch/dam/jcr:9826f155-6da5-48bd-9660-5793682fa372/Merkblatt_Seminararbeit_2021.pdf

[8] S. Branco and C. Niklaus, "Kontextbasierte Erkennung von Zitierungen in rechtlichen Dokumenten: Aufgabenstellung zur Bachelorarbeit im Herbstsemester 2024," Jul. 05, 2024, *Swisslex - Schweizerische Juristische Datenbank AG.*

[9] "SG 291.100 - Advokaturgesetz," Jul. 01, 2020, *Kanton Basel-Stadt.* Accessed: Dec. 14, 2024. [Online]. Available: https://www.gesetzessammlung.bs.ch/app/de/texts_of_law/291.100

[10] A. A. Awan, "What is Named Entity Recognition (NER)? Methods, Use Cases, and Challenges," Sep. 13, 2023, *DataCamp.* Accessed: Sep. 26, 2024. [Online]. Available: https://www.datacamp.com/blog/what-is-named-entity-recognition-ner

[11] J. Li, A. Sun, J. Han, and C. Li, "A Survey on Deep Learning for Named Entity Recognition." doi: 10.48550/arXiv.1812.09449.

[12] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed. Stanford University, 2024. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[13] M. Martínez-González, P. de la Fuente, and D.-J. Vicente, "Reference Extraction and Resolution for Legal Texts," *Pattern Recognition and Machine Intelligence*, pp. 218–221, 2005, doi: 10.1007/11590316_29.

[14] Y. Sari, M. F. Hassan, and N. Zamin, "Rule-based pattern extractor and named entity recognition: A hybrid approach," Sep. 02, 2010. doi: 10.1109/ITSIM.2010.5561392.

[15] A. Gheewala, C. Turner, and J.-R. de Maistre, "Automatic Extraction of Legal Citations using Natural Language Processing," *Proceedings of the 15th International Conference on Web Information Systems and Technologies (WEBIST 2019)*, pp. 202–209, May 2019, doi: 10.5220/0008052702020209.

[16] D. Mengliev, V. Barakhnin, M. Atakhanov, B. Ibragimov, M. Eshkulov, and B. Saidov, "Developing Rule-Based and Gazetteer Lists for Named Entity Recognition in Uzbek Language: Geographical Names," Dec. 13, 2023. doi: 10.1109/APEIE59731.2023.10347697.

[17] C. Dozier, R. Kondadadi, M. Light, A. Vachher, S. Veeramachaneni, and R. Wudali, "Named Entity Recognition and Resolution in Legal Text," *Semantic Processing of Legal Texts*, pp. 27–43, 2010, doi: 10.1007/978-3-642-12837-0_2.

[18] S. Morwal, N. Jahan, and D. Chopra, "Named Entity Recognition using Hidden Markov Model (HMM)," *International Journal on Natural Language Computing (IJNLC)*, vol. 1, Dec. 2012, doi: 10.5121/IJNLC.2012.1402.

[19] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning*. Springer, 2023, pp. 135–136.

[20] H. Yamada, T. Kudo, and Y. Matsumoto, "Japanese Named Entity Extraction Using Support Vector Machines," 2001, *In Transactions of IPSJ*.

[21] S. Saha and A. Ekbal, "Combining multiple classifiers using vote based classifier ensemble technique for named entity recognition," *Data & Knowledge Engineering*, vol. 85, pp. 15–39, May 2013, doi: 10.1016/j.datak.2012.06.003.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org/

[23] H. Xu and B. Hu, "Legal Text Recognition Using LSTM-CRF Deep Learning Model." doi: 10.1155/2022/9933929.

[24] E. Leitner, Rehm Georg Rehm, and J. Moreno-Schneider, "Fine-Grained Named Entity Recognition in Legal Documents," *Semantic Systems. The Power of AI and Knowledge Graphs*, pp. 272–287, Nov. 2019, doi: 10.1007/978-3-030-33220-4_20.

[25] A. Vaswani *et al.*, "Attention Is All You Need," Dec. 06, 2017. doi: 10.48550/arXiv.1706.03762.

[26] L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers*. O'Reilly Media, Inc., 2022.

[27] "Generative pre-trained transformer," Sep. 22, 2024, *Wikipedia*. Accessed: Sep. 27, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Generative_pre-trained_transformer

[28] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," Jun. 11, 2018, *OpenAI*. Accessed: Sep. 27, 2024. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[29] S. Wang *et al.*, "GPT-NER: Named Entity Recognition via Large Language Models," Oct. 07, 2023. doi: 10.48550/arXiv.2304.10428.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Oct. 11, 2018. doi: 10.48550/arXiv.1810.04805.

[31] B. Muller, "BERT 101 🤗 State Of The Art NLP Model Explained," Mar. 02, 2022, *Hugging Face*. Accessed: Sep. 30, 2024. [Online]. Available: https://huggingface.co/blog/bert-101

[32] H. Darji, J. Mitrović, and M. Granitzer, "German BERT Model for Legal Named Entity Recognition," *Proceedings of the 15th International Conference on Agents and Artificial Intelligence*, 2023, doi: 10.5220/0011749400003393.

[33] "LEGAL-BERT: The Muppets straight out of Law School," *Hugging Face.* Accessed: Jan. 01, 2025. [Online]. Available: https://huggingface.co/nlpaueb/legal-bert-base-uncased

[34] N. Guha *et al.*, "LegalBench: A Collaboratively Built Benchmark for Measuring Legal Reasoning in Large Language Models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 44123–44279, 2023, doi: 10.48550/arXiv.2308.11462.

[35] "Joel Nikluas." Accessed: Jan. 01, 2025. [Online]. Available: https://niklaus.ai/

[36] "Legal LMs," *Hugging Face.* Accessed: Jan. 01, 2025. [Online]. Available: https://huggingface.co/collections/joelniklaus/legallms-65303ccfc2f20ed637f17cb6

[37] J. Vamvas, J. Graën, and R. Sennrich, "SwissBERT: The Multilingual Language Model for Switzerland," *Proceedings of the 8th edition of the Swiss Text Analytics Conference*, pp. 54–69, 2023, Accessed: Jan. 01, 2025. [Online]. Available: https://aclanthology.org/2023.swisstext-1.6/

[38] U. Zaratiana, N. Tomeh, P. Holat, and T. Charnois, "GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer," Nov. 14, 2023. doi: 10.48550/arXiv.2311.08526.

[39] "Dataset Card for Wikipedia," *Wikimedia Foundation.* [Online]. Available: https://huggingface.co/datasets/legacy-datasets/wikipedia

[40] J. Alammar and M. Grootendorst, *Hands-On Large Language Models.* O'Reilly Media, Inc., 2024.

[41] "joelniklaus/legal-xlm-roberta-large," *Hugging Face.* Accessed: Jan. 01, 2025. [Online]. Available: https://huggingface.co/joelniklaus/legal-xlm-roberta-large

[42] M. Veton, M. Stürmer, Illias Chalkidis, D. E. Ho, and J. Niklaus, "MultiLegalPile: A 689GB Multilingual Legal Corpus," May 19, 2024, *arXiv.* doi: 10.48550/arXiv.2306.02069.

[43] IBM, "Zero and Few shot named entity & relationships recognition."

[44] "GPT-4o," *OpenAI.* Accessed: Dec. 26, 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[45] M. Goel *et al.*, "Deep Learning Based Named Entity Recognition Models for Recipes," 2024. [Online]. Available: https://arxiv.org/abs/2402.17447

[46] "Token Classification," *Hugging Face.* [Online]. Available: https://huggingface.co/docs/transformers/en/tasks/token_classification

[47] V. K. Keloth *et al.*, "Advancing entity recognition in biomedicine via instruction tuning of large language models," Apr. 04, 2024, *Bioinformatics.* doi: 10.1093/bioinformatics/btae163.

[48] C. Bucila, R. Caruana, and A. Niculescu-Mizil, "Model Compression," *KDD*, 2006, doi: 10.1145/1150402.1150464.

[49] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," Mar. 09, 2015. doi: 10.48550/arXiv.1503.02531.

[50] X. Xu *et al.*, "A Survey on Knowledge Distillation of Large Language Models," Feb. 20, 2024. doi: 10.48550/arXiv.2402.13116.

[51] C.-Y. Hsieh *et al.*, "Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes," May 03, 2023. doi: 10.48550/arXiv.2305.02301.

[52] A. Shirgaonkar, N. Pandey, N. C. Abay, T. Aktas, and V. Aski, "KNOWLEDGE DISTILLATION USING FRONTIER OPEN-SOURCE LLMS: GENERALIZABILITY AND THE ROLE OF SYNTHETIC DATA," Oct. 24, 2024, *arXiv*. doi: 10.48550/arXiv.2410.18588.

[53] "Synthetic data: save money, time and carbon with open source," *Hugging Face*. Accessed: Dec. 30, 2024. [Online]. Available: https://huggingface.co/blog/synthetic-data-save-costs

[54] T. Wu, M. Terry, and C. J. Cai, "AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts," Mar. 17, 2022. doi: 10.48550/arXiv.2110.01691.

[55] A. S. Kwak, C. T. Morrison, D. E. Bambauer, and M. Surdeanu, "Classify First, and Then Extract: Prompt Chaining Technique for Information Extraction," *Proceedings of the Natural Legal Language Processing Workshop 2024*, pp. 303–317, Nov. 2024, [Online]. Available: https://aclanthology.org/2024.nllp-1.25.pdf

[56] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications," Mar. 12, 2020, *arXiv*. doi: 10.48550/arXiv.2003.05689.

[57] A. Rakhecha, "Understanding Learning Rate," Jun. 28, 2019, *Towards Data Science*. Accessed: Nov. 11, 2024. [Online]. Available: https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de

[58] "Trainer," *Hugging Face*. Accessed: Nov. 11, 2024. [Online]. Available: https://huggingface.co/docs/transformers/v4.46.2/en/main_classes/trainer#transformers.TrainingArguments

[59] D. Vasani, "This thing called Weight Decay," Apr. 29, 2019, *Towards Data Science*. Accessed: Nov. 14, 2024. [Online]. Available: https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab

[60] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," Nov. 14, 2017, *arXiv*. doi: 10.48550/arXiv.1711.05101.

[61] "Batch Size," *Ultralytics Inc.* Accessed: Nov. 11, 2024. [Online]. Available: https://www.ultralytics.com/glossary/batch-size

[62] Devansh, "How does Batch Size impact your model learning," Jan. 17, 2022, *Geek Culture*. Accessed: Nov. 08, 2024. [Online]. Available: https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa

[63] M. Cacic, "What are Fine-tuning Hyperparameters and How to Set Them Just Right 👌😩," Aug. 14, 2023, *Entry Point AI Inc.* Accessed: Nov. 11, 2024. [Online]. Available: https://www.entrypointai.com/blog/fine-tuning-hyperparameters/

[64] Xuan_Ren, "Why is the default batch size set to 1 for fine-tuning the ChatGPT Turbo model?," Nov. 2023, *OpenAI*. Accessed: Nov. 11, 2024. [Online]. Available: https://community.openai.com/t/why-is-the-default-batch-size-set-to-1-for-fine-tuning-the-chatgpt-turbo-model/513129

[65] "ExOpenAI.FineTunes.create_fine_tune," *hexdocs*. Accessed: Nov. 11, 2024. [Online]. Available: https://hexdocs.pm/ex_openai/ExOpenAI.FineTunes-function-create_fine_tune.html

[66] A. Thakur, "What's the Optimal Batch Size to Train a Neural Network?," *Weights & Biases*. Accessed: Nov. 11, 2024. [Online]. Available: https://wandb.ai/ayush-thakur/dl-question-bank/reports/What-s-the-Optimal-Batch-Size-to-Train-a-Neural-Network---VmlldzoyMDkyNDU

[67] P. Radhakrishnan, "What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?," Aug. 09, 2017, *Towards Data Science*. Accessed: Nov. 11, 2024. [Online].

Available: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

[68] "Universal Named Entity Recognition," Jan. 19, 2024. doi: 10.48550/arXiv.2308.03279.

[69] "Named Entity Evaluation as in SemEval 2013 task 9.1," *GitHub*. Accessed: Dec. 30, 2024. [Online]. Available: https://github.com/davidsbatista/NER-Evaluation

[70] J. Fu, P. Liu, and G. Neubig, "Interpretable Multi-dataset Evaluation for Named Entity Recognition," Nov. 13, 2020. doi: 10.48550/arXiv.2011.06854.

[71] "C4 model." Accessed: Jan. 04, 2025. [Online]. Available: https://c4model.com/

[72] "Y-Statements," *Olaf Zimmermann*. Accessed: Jan. 01, 2025. [Online]. Available: https://socadk.github.io/design-practice-repository/artifact-templates/DPR-ArchitecturalDecisionRecordYForm.html

[73] "NVIDIA DGX H100," Dec. 30, 2024, *NVIDIA*. [Online]. Available: https://resources.nvidia.com/en-us-dgx-systems/ai-enterprise-dgx?ncid=no-ncid&xs=489753

[74] "Welcome to vLLM!," *vLLM*. Accessed: Dec. 30, 2024. [Online]. Available: https://docs.vllm.ai/

[75] "Python," *Python Software Foundation*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.python.org/

[76] "PyTorch," *PyTorch*. Accessed: Dec. 30, 2024. [Online]. Available: https://pytorch.org/

[77] "Hugging Face," *Hugging Face*. Accessed: Dec. 30, 2024. [Online]. Available: https://huggingface.co/

[78] "FastAPI," *FastAPI*. Accessed: Dec. 30, 2024. [Online]. Available: https://fastapi.tiangolo.com/

[79] ".NET," *Microsoft*. Accessed: Dec. 30, 2024. [Online]. Available: https://dotnet.microsoft.com/

[80] "Blazor," *Microsoft*. Accessed: Dec. 30, 2024. [Online]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

[81] "log4net," *Apache Software Foundation*. Accessed: Jan. 08, 2025. [Online]. Available: https://logging.apache.org/log4net/

[82] "Elasticsearch," *Elastic*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.elastic.co/elasticsearch

[83] "SQL Server," *Microsoft*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.microsoft.com/en-us/sql-server

[84] "PyCharm," *JetBrains*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.jetbrains.com/pycharm/

[85] "Visual Studio," *Microsoft*. Accessed: Dec. 30, 2024. [Online]. Available: https://visualstudio.microsoft.com/

[86] "Rider," *JetBrains*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.jetbrains.com/rider/

[87] "DataGrip," *JetBrains*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.jetbrains.com/datagrip/

[88] "Visual Studio Code," *Microsoft*. Accessed: Dec. 30, 2024. [Online]. Available: https://code.visualstudio.com/

[89] "Azure DevOps," *Microsoft*. Accessed: Dec. 30, 2024. [Online]. Available: https://azure.microsoft.com/en-us/products/devops

[90] "JupyterLab," *Jupyter*. Accessed: Dec. 30, 2024. [Online]. Available: https://jupyterlab.readthedocs.io/en/latest/

[91] "MLflow." Accessed: Dec. 27, 2024. [Online]. Available: https://mlflow.org/

[92] "Prodigy." [Online]. Available: https://prodi.gy/

[93] "Typst," *Typst*. Accessed: Dec. 30, 2024. [Online]. Available: https://typst.app/

[94] "ChatGPT," *OpenAI*. Accessed: Dec. 30, 2024. [Online]. Available: https://chatgpt.com/

[95] "GitHub Copilot," *GitHub*. Accessed: Dec. 30, 2024. [Online]. Available: https://github.com/features/copilot

[96] "jq is a lightweight and flexible command-line JSON processor.," *jqlang*. Accessed: Dec. 30, 2024. [Online]. Available: https://jqlang.github.io/jq/

[97] "Microsoft Teams," *Microsoft*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software

[98] "Threema," *Threema*. Accessed: Dec. 30, 2024. [Online]. Available: https://threema.ch/

[99] "Miro," *Miro*. Accessed: Dec. 30, 2024. [Online]. Available: https://miro.com/

[100] "Jira," *Atlassian*. Accessed: Dec. 30, 2024. [Online]. Available: https://www.atlassian.com/software/jira

[101] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," Oct. 02, 2019. doi: 10.48550/arXiv.1910.01108.

[102] P. He, J. Gao, and W. Chen, "DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing," Nov. 18, 2021. doi: 10.48550/arXiv.2111.09543.

[103] A. Conneau *et al.*, "Unsupervised Cross-lingual Representation Learning at Scale," Nov. 05, 2019. doi: 10.48550/arXiv.1911.02116.

[104] "unsloth/Llama-3.2-1B-Instruct-bnb-4bit," *Hugging Face*. Accessed: Dec. 30, 2024. [Online]. Available: https://huggingface.co/unsloth/Llama-3.2-1B-Instruct-bnb-4bit

[105] "unsloth/Llama-3.2-3B-Instruct-bnb-4bit," *Hugging Face*. Accessed: Dec. 30, 2024. [Online]. Available: https://huggingface.co/unsloth/Llama-3.2-3B-Instruct-bnb-4bit

[106] "unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit," *Hugging Face*. Accessed: Dec. 26, 2024. [Online]. Available: https://huggingface.co/unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit/tree/main

[107] "unsloth/Llama-3.3-70B-Instruct-bnb-4bit," *Hugging Face*. Accessed: Dec. 30, 2024. [Online]. Available: https://huggingface.co/unsloth/Llama-3.3-70B-Instruct-bnb-4bit

[108] Y. Chen *et al.*, "LONGLORA: EFFICIENT FINE-TUNING OF LONG- CONTEXT LARGE LANGUAGE MODELS," Sep. 21, 2023. doi: 10.48550/arXiv.2309.12307.

[109] S. N. Kunnath, D. Pride, and P. Knoth, "Dynamic Context Extraction for Citation Classification," *The 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing*, 2022.

[110] "MinHash," Nov. 14, 2024, *Wikipedia*. Accessed: Dec. 20, 2024. [Online]. Available: https://en.wikipedia.org/wiki/MinHash

[111] G. Penedo, H. Kydlíček, A. Cappelli, M. Sasko, and T. Wolf, "DataTrove: large scale data processing," *GitHub*. Accessed: Dec. 20, 2024. [Online]. Available: https://github.com/huggingface/datatrove

[112] "Hugging Face Models," *Hugging Face*. Accessed: Nov. 26, 2024. [Online]. Available: https://huggingface.co/models

[113] "LoRA," *Hugging Face*. Accessed: Dec. 26, 2024. [Online]. Available: https://huggingface.co/docs/peft/main/en/developer_guides/lora

[114] "PEFT," *Hugging Face*. Accessed: Dec. 26, 2024. [Online]. Available: https://huggingface.co/docs/peft/main/en/index

[115] E. J. Hu *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models," Jun. 17, 2021. doi: 10.48550/arXiv.2106.09685.

[116] "gliner/training/trainer.py." Accessed: Jan. 10, 2025. [Online]. Available: https://github.com/urchade/GLiNER/blob/main/gliner/training/trainer.py

[117] "meta-llama/Llama-3.1-8B-Instruct," *Hugging Face*. Accessed: Dec. 26, 2024. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct/tree/main

[118] E. F. T. K. Sang and S. Buchholz, "Introduction to the CoNLL-2000 Shared Task: Chunking," Sep. 18, 2000. doi: 10.48550/arXiv.cs/0009008.

[119] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, updated1 ed. O'Reilly Media, Inc., 2009. [Online]. Available: https://www.nltk.org/book/

[120] "Train on data without entities," *AnnaKholkina*. Accessed: Jan. 05, 2025. [Online]. Available: https://github.com/urchade/GLiNER/issues/139

[121] "Ray Tune: Hyperparameter Tuning," 2024, *The Ray Team*. Accessed: Dec. 23, 2024. [Online]. Available: https://docs.ray.io/en/latest/tune/index.html

[122] G. Jawahar, B. Sagot, and D. Seddah, "What Does BERT Learn about the Structure of Language?," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, doi: 10.18653/v1/P19-1356.

[123] H. Touvron *et al.*, "LLaMA: Open and Efficient Foundation Language Models," Feb. 27, 2023. doi: 10.48550/arXiv.2302.13971.

[124] "vLLM Documentation." Accessed: Dec. 29, 2024. [Online]. Available: https://docs.vllm.ai/

[125] "OpenAI Compatible Server," *vLLM Team*. Accessed: Dec. 29, 2024. [Online]. Available: https://docs.vllm.ai/en/latest/serving/openai_compatible_server.html

[126] "Introduction to Automatic Prefix Caching," *vLLM Team*. Accessed: Dec. 30, 2024. [Online]. Available: https://docs.vllm.ai/en/v0.5.5/automatic_prefix_caching/apc.html

[127] "Performance and Tuning," *vLLM Team*. Accessed: Dec. 30, 2024. [Online]. Available: https://docs.vllm.ai/en/v0.4.2/models/performance.html

[128] "Engine Arguments," *vLLM Team*. Accessed: Dec. 30, 2024. [Online]. Available: https://docs.vllm.ai/en/latest/usage/engine_args.html

[129] "CI Benchmark," *vLLM Team*. Accessed: Dec. 30, 2024. [Online]. Available: https://perf.vllm.ai/

[130] "OpenAPI Specification," *OpenAPI Initiative*. Accessed: Jan. 10, 2025. [Online]. Available: https://swagger.io/specification/

[131] "Transformers," *Hugging Face*. Accessed: Dec. 30, 2024. [Online]. Available: https://huggingface.co/docs/transformers/index

[132] "unittest — Unit testing framework," *Python Software Foundation*. Accessed: Dec. 30, 2024. [Online]. Available: https://docs.python.org/3/library/unittest.html

[133] B. Warner *et al.*, "Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference," Dec. 18, 2024. doi: 10.48550/arXiv.2412.13663.

[134] "Nvidia DGX," Sep. 13, 2024, *Wikipedia*. Accessed: Sep. 20, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Nvidia_DGX