Armin Weiß, BSc.

MIGRATING UNANCHORED TEXT TAGS ACROSS PDF VERSIONS

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme:
Computer Science

submitted to
Graz University of Technology

Supervisor
Dipl.-Ing. Dr.techn. BSc, Stefan Lengauer
Institute of Visual Computing

Graz, February 2026

Abstract (English)

Researchers in the field of visualization are constantly on the lookout for innovative systems to convey knowledge on different topics more efficiently. One such type of system being researched is interactive exploration systems, which employ a wide set of different visualization methods to let consumers explore a large amount of knowledge available in a domain more easily than linear reading. The data required by those employed visualization methods can either be provided by automatic natural language processing (NLP) methods or with manual assistance by domain experts who annotate or classify information contained in a document or corpus. One problem that arises with manual annotations by domain experts on documents is that documents do not necessarily stay static over their entire life cycle. Evolving versions of the same document may change, move, or even delete text and other content that was annotated previously. Manual verification and adjustment of the integrity of existing annotations in a changed document is a very cumbersome task for domain experts which raises the need for a tool that assists this process and automatizes it as much as possible. For this Master's thesis, such a tool was developed in order to assist with the migration of tags, annotating content with absent location information for PDF documents. We investigate the applicability of the approach one one selected use case of the health domain.

Abstract (German)

Wissenschaftler im Feld der Visualisierung sind konstant auf der Suche nach innovativen Systemen, die in der Lage sind Wissen über verschiedene Themen effizienter zu übermitteln. Systeme zur interaktiven Exploration sind eine Art solcher Systeme, welche eine Vielzahl an unterschiedlichen Visualisierungsmethoden einsetzen um Nutzern eine größere Menge an Wissen einer Domäne einfacher zugänglich zu machen als durch lineares Lesen. Die zur Visualisierung notwendigen Daten können entweder durch automatische Natural Language Processing (NLP) Methoden erfolgen oder direkt durch Domänenexperten bereitgestellt werden, welche die Information in einem Dokument oder Korpus klassifizieren. Ein Problem, welches durch manuelle Annotationen von Domänenexperten entsteht ist, dass Dokumente über ihren gesamten Lebenszyklus nicht unbedingt unverändert bleiben. Sich weiterentwickelnde Versionen des selben Dokuments können Text verändern, verschieben oder sogar komplett entfernen, welcher in einer vorhergehenden Version annotiert wurde. Die manuelle Verifikation der Integrität und die Anpassung von existierenden Annotationen in einem veränderten Dokument ist eine sehr arbeitsintensive Aufgabe für Domänenexperten. Hieraus resultiert der Bedarf für eine Applikation, welche diesen Prozess unterstützt und so weit automatisiert wie möglich. Für diese Masterarbeit wurde ein solches Tool entwickelt um bei der Migration von Tags, welche Text ohne Positionsinformation in PDF Dokumenten annotieren zu migrieren. Wir untersuchen die Anwendbarkeit unserer Vorgehensweise an einem ausgewählten Use Case aus dem Gesundheitsbereich.

Contents

1 Introduction

In this chapter we will discuss relevant background information that is required to understand the problem at hand in more detail and our core objectives.

1.1 Background and Motivation

Within the last years, a research team from the Institute of Visual Computing of Graz University of Technology in collaboration with other institutions and universities has started to build an interactive Consumer Health Information System (CHIS) within the project 'Human-Centered Interactive Adaptive Visual Approaches in High-Quality Health Information' (A$^+$CHIS[1] ; Grant No. FG 11-B) which was funded by the Austrian Science Fund (FWF). This system is meant to enable research on the advantages a combination of close and distant reading approaches (in the sense of Moretti[2], 2000) and potential automatic adaptations of such visualizations has over typical linear consumption of readable Consumer Information System (CIS) resources on a complex topic.

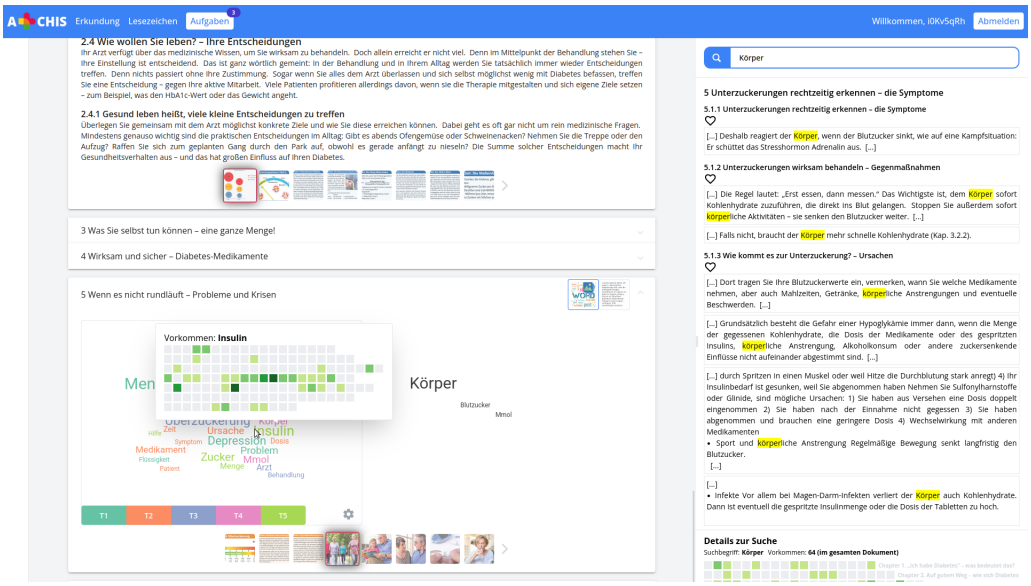


Figure 1.1: GUI of the A$^+$CHIS Web Explorer - Source: [3]

Currently, the information that A$^+$CHIS provides is based on an established CHIS in the form of a

brochure[1] on Type 2 diabetes (T2DM) issued by the AOK[4] (a German public health insurer). The effectiveness of having a consumer extract required information from this interactive system compared to just reading the brochure has been evaluated to some extent by Lin et al. (2023) [5]. They showed promising results based on a small number of participants. The system, however, did not automatically adapt its visualizations due to any inferences of personalized customer needs at that point. Currently, research is being done on how to add this automatic adaptability to the system. Further, Lengauer et al. (2023) [6] elaborated on their initial prototype and defined a process landscape to infer different cognitive states a consumer can have while interacting with the system. The inference of that cognitive state is based on the user's input and the context in which they are currently browsing the provided information. Based on such inferred cognitive states the system should then be able to automatically change its visualizations to fit consumer needs in the best possible way.

The visualization methods of A$^+$CHIS depend on annotations in the PDF brochure made by medical professionals using the tools MAXQDA[7] and Qualcoder[8]. Information from the PDF is extracted using PDFBox[9] and after further preprocessing, this data is then fed into the A$^+$CHIS application server which is implemented as a Python Django web application. The annotations generated by the aforementioned tools are provided separately from the document as a *.csv* file.

The PDF brochure however, changes over its life cycle and the current information fed into the A$^+$CHIS system still refers to a brochure version that was released many years ago. Text and other assets were moved around, refactored, or even removed from the document over time. Therefore, if the project was to stay up to date on newest brochure versions, the necessity arises for the professionals who initially annotated the document to re-do their work every time the document changes. The purpose of the application that was developed for this thesis is to assist this process by automatizing most of the manual labor.

1.2 Problem Statement And Objectives

The core problem is a situation where text that is annotated, no longer exists or is no longer equal to the text that a tag was initially anchored to. Bernheim Brush et al. (2001) [10] discusses this specific problem and Cadiz et al. (2000) [11] coined this problem *Annotation Orphaning* when noting that this problem was the main reason, their study participants quit using the (in the study) evaluated Microsoft Office feature called Web Discussions which allowed for annotation of website content in a collaborative way.

It is not possible to obtain the sources that were used to generate the PDF file version of the CHIS that A$^+$CHIS is based on. Therefore, a comparison of text between the source and target documents has to be done based on the provided PDF files. The problem outlined above is an even larger one for this context because a PDF file mostly only represents its text content as drawing instructions and does not store any metadata associated with its texts. As an example, a PDF file does not store that a section header is meant to be just that, but it rather only stores that the text of a section header is larger than some other text that is drawn beneath it. Also, it is not a trivial problem to determine which texts on one or multiple pages belongs to each other.

The domain experts for a CIS provide a tag tree where each tag represents a topic. Such topics can then have subtopics which are represented by other tags. These domain experts annotated different text passages of a document with tags from this tag tree. Within this thesis, the terms *annotation* and *tag* as well as *annotating* and *tagging* will be used interchangeably. A$^+$CHIS uses a custom ZIP-based file format has extensive text annotation capabilities. This *.apchis* file format provides the content of a respective CIS document in a *.csv* file on which content this thesis primarily revolves around. Each row within this file represents a chapter and contains a list of all taggable sentences that occur in a given chapter as well as a list of tag-identifiers referencing one or more sentence-indices to indicate which sentences of the chapter are tagged by what tag. Although in newer versions of the A$^+$CHIS format, each chapter-row also contains start page numbers, there is no other information stored that links the enumerated sentences of a chapter to its exact location within the document. Also, the stored sentences are sanitized and do not absolutely match the actual text in the document token by token. For example, hyphens that occur on line breaks are not stored. Some other tokens are replaced with different ones like bullet points. This means, in order to visualize annotated sentences within the desired tool, the missing exact token locations of each annotated sentence have to be re-established first. This in turn also means, that a *.csv* file with the same structure has to be generated for the updated version of the brochure where additional chapters may have been added while other ones may have been moved, removed or changed.

- Thumbnail image of the documents cover
- Extracted figures as images
- *.csv* file containing meta-information on the extracted figures
- *.json* file containing the configuration
- *.pdf* document of the brochure in a given version
- *.json* file containing a list of topics and associated terms with term-weights
- *.csv* file providing the content of the document as a list of chapters (See Table 1.1)

Table 1.1: This table shows the columns of the content .csv file within the .apchis file format of version 1.4.3.

Column	Data Type	Example
type	string	text:ke
chapter_number	string	1.2.1
chapter_title	string	Kurz erklärt: Diabetes Typ 1 und 2
content	list[string]	['This is a sentence', 'This is another sentence', ...]
annotations	dictionary[string, list[int]]	{'43': [0, 1, 2, 3, 4], '45': [0, 1, 2, 3, 4]}
page	list[int]	[12]

As can be seen in Table 1.1, no concrete text locations are provided for the listed content sentences. However, as also shown, a list of pages is provided in the most current version of the *.apchis* file format which was not the case for the version that the project started with (1.4.0). None of the other file entries present within the *.apchis* file provide any text location information either. This is because, at the time the currently used *.apchis* file content for the system was generated, it was not known that more concrete text locations would ever be required. One goal of this project is to have a redundancy in the form of an additional file entry within the *.apchis* file format that provides this missing information in the next file version that is upgraded to the most current document version.

To conclude, a tool to assist with the migration of tagged content of an *.apchis* file needs to tackle the following main steps to achieve a useful editor:

1. Extraction of all coherent texts scattered over complex page layouts in both the source and target document
2. Splitting the content of coherent text clusters into processable sentences
3. Inferring the exact location for all pre-known sentences of each chapter in the source document and enrichment of the pre-existing *.apchis* file with these locations **(Enrichment Step)**
4. Establishing the chapter hierarchy of the target document from its content **(Target Chapter Inference Step)**
5. Extracting all individual sentences of the target document and connecting them to the appropriate inferred chapters
6. Automatized matching of sentences between the source and target document **(Migration Step)**
7. Implementation of a GUI that visualizes automatically inferred information and provides the tools required by the user to intervene manually where necessary

In the following chapters we will first discuss related work and fundamental concepts. After that, the implementation of our application, further refered to as A$^+$CHIS Migrator (A$^+$CHIS^M) will be described in more detail and an overview of our architecture and pipeline of different steps will be presented. Finally, the performance of our approach will be evaluated based on a real use case of the A$^+$CHIS application.

Note that this thesis contains multiple figures depicting different screenshots of the GUI views of A$^+$CHIS^M where a user can make manual corrections. All of these figures show content from the older or newer version of the AOK-brochure [12] which will not be explicitly referenced per figure.

2 Related Work

No recent literature focusing on the very specific problem of annotation orphaning seems to exist. Especially in the context of PDF files. No tools were found that support smart handling or assistance to deal with the specific issue at hand well when it comes to general purpose PDF files with complex page layouts and the way text-content is provided for A$^+$CHIS.

2.1 PDF Difference Detection

As discussed further in Section 3.2 tools that operate on default PDF annotations are not helpful for the problem at hand because none of the annotations are provided as default PDF annotations. Instead they are provided outside of the document because default PDF annotations cannot hold all the information that would be required without customizing the documents internals.

Recently, some commercial projects like Draftable[13], DiffitDoc[14] or V7Labs Agent[15] were established that can infer the differences between two different versions of the same PDF file even with significant layout changes. However, all discovered commercial projects like these seem to focus primarily on purely text-based content and not more complex page layouts like the A$^+$CHIS project has to work with, where text may frequently be interrupted by boxes of unrelated text blocks or images. Naive extraction of all texts of a PDF file's pages with layouts such as this will inevitably mix words of clusters that do not belong to each other. Moreover, the primary purpose of those projects seems to be to visualize the changes between two PDF file versions for inspection instead of providing these differences in metadata formats which would enable further processing in order to solve the problem at hand.

2.2 PDF Content Extraction

There also does not seem to be any recently published literature that is directly dedicated to the annotation orphaning problem within PDF files. Generally, research within the last decade on the topic of information extraction from PDF files, seems to have primarily targeted scientific literature over general purpose documents. A few recent publications however, combine a mixture of different technology pipelines to achieve solid results for such general purpose documents. Adjetey et al. (2021) [16] showed that the combination of Optical character recognition (OCR)-based text extraction with string distance measures like the Levenshtein-Distance is a valid strategy to find which document of a set of documents contains specific text snippets contained in query images. For documents used in A$^+$CHIS, the sentences of the source document are pre-known. It therefore stands to reason that the same approach is viable for the problem at hand because the respective document could be split into multiple sub documents. Chen et al. (2025) [17] showed that with a combination of OCR and text-only Large language models (LLMs), a set of predefined variables could be extracted from documents with inconsistent layouts by extracting

information like dates and amounts from shipping-invoices. Moreira-Filho et al. (2025) [18] compared a text-based mode using layout detection, feeding into an LLM with an image-based approach where PDF pages are converted into images and are then fed into multimodal LLMs. They showed that the image-based approach significantly outperformed the text-based one on general purpose documents with inconsistent layout when extracting pre-defined variables. Most desired information could be extracted from toxicology safety data sheets with this image-based approach. One problem that was not explicitly mentioned in any of this literature is the cross-page sentence continuation problem where on multiple occasions within a document, a sentence either continues at an unpredictable location on the following page or even skips a page in between. We reason that the mentioned image-based multimodal LLM approach would be able to solve this problem by providing a sufficiently long window of pages from the document to analyze to the LLM for each extraction step.

Ingemarsson and Persson (2025) [19] evaluated different techniques for parsing PDF files for further use with LLMs on different document layouts. They come to the conclusion that multimodal LLM-based techniques generally work best for a wide range of documents with complex layouts that contain not only text but also tables and images as compared to rule-based approaches. Documents with many images however, should still additionally employ OCR-based methods.

We assume, approaches using multi-modal LLMs could assist or are potentially superior in some aspects for identifying the locations of pre-known sentences within the source document compared to our approach. Especially on cases where a single sentence or paragraph is distributed over multiple pages while being interrupted by other content, employing such a LLM could potentially find sentences automatically that our approach cannot. Future work could evaluate if multi-modal LLMs are capable to extract exact page coordinates and dimensions of text passages which is required for our $A^{+}CHIS^{M}$.

2.3 Annotation orphaning

The situation in which text moves, disappears or is changed that has annotations or tags anchored to it, is known as annotation orphaning. Bernheim Brush et al. (2001) [10] took a look at some tools and how they deal with this problem generally (not necessarily for PDF files). The typical approaches they found were: restriction of annotation positioning in the first place, silently discarding orphaned annotations, and algorithmic approaches to reassign the position of an orphaned annotation. Phelps & Wilensky (2000) [20] suggest to be redundant when storing initial annotation locations. Namely, one should store a unique identifier of the containing structure, tree walk information (how to traverse the document hierarchy to arrive at the location), and the direct context surrounding the annotation. A unique identifier is not really possible when dealing with PDF documents but tree walk information is at least roughly possible since content streams are associated with pages and one could build a pseudo-DOM hierarchy of the content of such a page to simulate tree traversal. When it comes to the surrounding context of a text annotation, one can take a small portion of words around the highlight and store them as anchor information. The smaller the considered context around the annotation, the more likely it becomes to encounter conflicts when dealing with large documents. This is also why redundant storage of the other kinds of anchor information makes sense. In the case of PDF, this does not necessarily only have to be text. As established earlier, most visible content that is displayed from a PDF file has an underlying drawing operation. One could also store neighboring content stream operations to the drawing operation of the text being annotated. Rather than storing tagging information directly inside the *.pdf* file, A^{+}CHIS instead relies on external files that carry additional required information as otherwise customized software would be required to extract the stored information again.

3 Fundamentals and related topics

In this section, fundamental concepts required to understand the problem and techniques used by A^{+}CHIS^{M} will be elaborated on. The In this chapter we will discuss relevant fundamental information and technologies which A^{+}CHIS^{M} employs to achieve it's automation.

3.1 The Structure of PDF files

How a PDF document is structured from a file perspective, is standardized by ISO 32000 [21]. The smallest building blocks that a PDF file consists of are called COS-objects which can be of different types that resemble both primitive and complex data types of most modern programming languages. At a very high level, the content of the document is structured into four different sections (see Figure 3.1). Those are the header, body, cross-reference table, and trailer which occur in this exact order. While the beginning and end of the header and body section are implicit based on their location and content, the beginning and end of the cross-reference table and trailer are defined by their respective keywords "xref" and "trailer". Most of the time the header only contains one comment to define the PDF version that the file adheres to and another comment containing a sequence of high-value ASCII characters to fool software into recognizing it as binary. The body contains the actual content of the entire document. The content is a mix of drawing directives, textual content, binary information in the form of stream objects (for example to represent the content of an image), annotations, and more. Content can either be directly stated or indirectly referenced by pointing at objects that occur later in the body section with the exception of stream objects which always have to be referenced. The cross-reference table section provides the binary offset for all such references from the beginning of the file. The trailer section states how many references are to be found in the cross-references table and references the documents catalog dictionary which is also called the root of the document. This root dictionary is the entry point from which the contents of the document can be navigated as it contains the reference to the page tree. The page tree is either a simple array of page objects or a balanced tree with "page" dictionary objects as leaf nodes and "pages" dictionary objects as non-leaf nodes (see Figure 3.2). Each page dictionary then states its drawing instructions, content references, and also annotations required in order to render the page [22].

A general understanding of how the PDF standard works, is beneficial in solving the problem of extracting structured text from an existing document. However, in-depth knowledge is not required to build sophisticated applications for such a task as many programming languages have access to third party libraries like PDFBox[9] that abstract most if not all low-level manipulations that deal with COS objects directly.

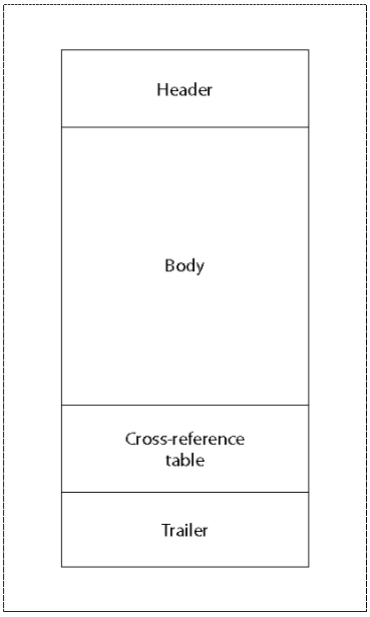


Figure 3.1: Top-level structure of a PDF document - Source: [22]

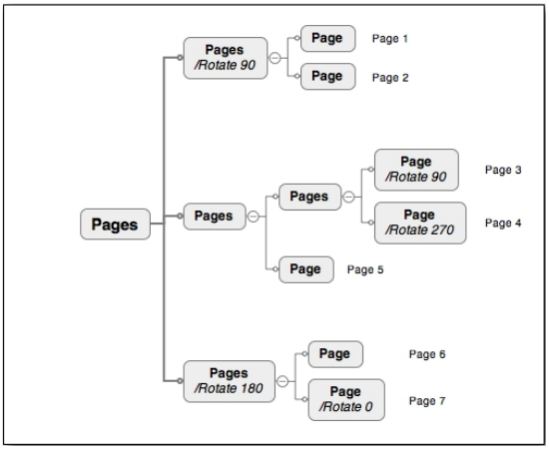


Figure 3.2: Page Tree of a PDF document - Source: [22]

3.2 A+CHIS Tags / Default PDF Annotations

Generally, annotations are additional types of objects that extend the document with visible content that the basic content standard does not provide. Typically annotations overlay the already present content to mark areas of interest and enhance their visual representation in some way. A simple annotation could be one of the many highlighting options for a text passage or an interactive (clickable) comment indicator that expands into a written comment that someone left. The many different kinds of annotations basically make the PDF file a canvas where annotations can be drawn on top of it. Audio, video, 3D Objects or even file attachments are possible to be maintained. While the basic content of a page is stored as a stream of graphic operators on that page, annotations are not drawn by such graphic operators but they are also stored on page level. They are stored as an array of annotations for the "Annots" key contained in the pages page-dictionary [22].

As an example, when looking at a standard highlight annotation in a PDF file, the storage would work as follows. The "Annots" dictionary of the page where the highlight is placed would contain a reference to some annotation object representing this highlight. Theoretically, the object could also be stored directly in the dictionary without being referenced but this is usually not the case. Instead, this annotation object would be stored somewhere else in the body section of the document and would look like Listing 3.1. As can be seen here, the annotation is not anchored to the text itself but is instead spanned up by one or multiple bounding boxes like a drawing operation. The Rect key defines an array that provides the bounding box for the entire highlight. In case the highlight just has a simple rectangular shape without any rotation, this bounding box is already sufficient to define what is highlighted. In case the highlighted text is rotated or spans over multiple lines where the highlighted text is not equally long in each line, the `QuadPoint` array is required. In this array, arbitrary multiples of eight space-separated number values can be stored which represent the corners of different bounding boxes which in combination define how the highlight should be drawn on the page. The `C` key represents the color of the highlight.

In the context of this thesis it is important to emphasize that the word tag(s) and annotation(s) is used interchangeably. In this thesis, the word annotation specifically does not refer to the annotations that are typically meant in the context of PDF documents. In the *.apchis* file format the original document that the information is based on remains untouched, meaning no additional content was added to the document. All tags or "annotations" are provided as a separate *.json* file representing the tag tree as a configuration file outside of the *.apchis* file while the information which sentences are tagged with what tag is maintained on the *.csv* file inside the *.apchis* file. To contrast, in the following it will be explained how default annotations work within the PDF standard.

Listing 3.1: Highlight annotation object in PDF Domain-specific language (DSL)

```
10 0 obj
<<
    /Type /Annot
    /Subtype /Highlight
    /Rect [ 250.5 270.0 340.0 282.5 ]
    /QuadPoints [ ... ] % optional
    /C [ 1.0 0.5 0.0 ]
>>
```

3.3 Sentence Comparison by String Similarity

In order to compare two pieces of text with trivial modifications, string similarity distance functions like the Levenshtein-Distance[23] or the Jaro-Winkler-Distance[24] can be used. The former measures the amount of character adjustments that have to be made in order to turn one sentence into another while the latter excels at character transpositions. While the Jaro-Winkler-Distance maps the comparison of two sentences in a range between zero and one directly, the comparison of sentences via the Levenshtein-Distance requires a mapping into that range such that the result can be compared against some threshold. For A+CHIS^{M} the result is calculated by subtracting the calculated Levenshtein-Distance of both sentences divided by the max sentence length from 1.0. In cases where one sentence is significantly longer than another, a performance optimization is made by A+CHIS^{M} which returns a similarity value of 0.0 right away instead of performing the expensive distance calculation for a case where it does not seem to make sense.

While the implementation of A$^+$CHIS^M makes use of a library implementation that abstracts the internals of the Levenshtein-Distance, Equation (3.1) shows the algorithm as defined by Wagner and Fisher (1974) [25]. The first and second input strings are denoted by `a` and `b` respectively while `m` and `n` denote their lengths. The matrix D is calculated according to the provided formula where each $D_{i,j}$ shows the minimum edit distance for transforming the prefix of the first string $a_{1..i}$ into the prefix of the second string $b_{1..j}$. $D_{m+1,n+1}$ is therefore the final resulting Levenshtein-Distance between the first and second input string.

$$D_{i,j} = \begin{cases} j, & \text{if } i = 0, \\ i, & \text{if } j = 0, \\ \min \begin{cases} D_{i-1,j} + 1, \\ D_{i,j-1} + 1, \\ D_{i-1,j-1} + \mathbf{1}_{a_i \neq b_j} \end{cases}, & \text{otherwise.} \end{cases} \tag{3.1}$$

3.4 Semantic Sentence Comparison by BERT embeddings

Sentences that changed only slightly by either having few words replaced or a small amount of additional words added are matchable with the aforementioned string similarity measures. In order to match greater changes to a sentence, a measure on the semantic level is required. While older rudimentary statistical approaches like Bag of words (BoW) or Term Frequency-Inverse Document Frequency (TF-IDF) can be used to compare sentences semantically to a minor extent, they do not suffice for the task at hand. For machine learning approaches, static word embedding models like Word2Vec by Mikolov et al. (2013) [26] and GloVe by Pennington et al. (2014) [27] are nowadays outperformed by contextual language models for sentence comparison tasks. In A$^+$CHIS^M we use BERT by Devlin et al. (2019) [28] for semantic text comparison. To compare two sentences, the embeddings for both sentences are calculated first. Generally embeddings are vectors. Each word piece of a sentence is assigned a vector and all those vectors of a sentence are then "pooled" into a single embedding (vector) that represents the sentence by different methods. For determining the embedding of each word piece, BERT considers both, the word pieces on the left and on the right side of it. For each word piece, it is determined how relevant each other word piece in the sentence is for its meaning. The resulting sentence embeddings are then vectors of the same length which can be compared to each other by some distance measure like the cosine similarity.

The model used by A$^+$CHIS^M is *bert-multi-cased-L-12-H-768-A-12*. Dissected, this means:

- multi-cased: multiple languages are supported.
- L-12: the computations to calculate the embeddings happen on twelve layers (larger BERT models with up to 24 layers exist).
- H-768: The embeddings have 768 dimensions.
- A-12: The number of attention heads (different ways of determining if a word piece is relevant for another one).

Within A$^+$CHIS^M we use the library easy-bert[29] which is publicly available and provides an API where a sentence can be passed into the model and the embedding vector is returned. After embeddings for both sentences are retrieved this way, only the cosine similarity has to be calculated, and the result which is a number between 0 and 1 has to be compared against a threshold to determine if the sentences are sufficiently similar semantically for the purposes of A$^+$CHIS^M.

4 System Architecture

A one step process to migrate all tags of one document version to another is not feasible as many edge cases arise which make manual intervention and corrections necessary at a very early point in the workflow. Due to high runtime durations of matching only the known sentences of the source document to its locations in the corresponding document, we decided that the entire processing pipeline that leads from the initial *.apchis* file to an *.apchis* file based on a newer document version of the brochure should be split up into multiple steps.

The application is therefore structured into three different steps where each of them first attempts to infer required information automatically and then provides their own GUI view for manual additions and corrections. Each of the steps requires different input files and produces different output files. As shown in Figure 4.1, the final step requires the output of two pre-processing steps to produce the updated *.apchis* file. Both pre-processing steps require layout information about either the source or target PDF document which is acquired by making use of a third party open source service which will be elaborated on later.

- **Enrichment Step:** Enriches the provided legacy *.apchis* filewith text location information. An additional *.json* file is added as an entry into the zipped format. This file is used to store additional location information for each chapter-sentence provided in the pre-existing annotation *.csv* file. For a given chapter ID and sentence index within that chapter, a list of rectangles associated with page numbers (in the following referred to as *page-rect*) which visually cover all text locations the sentence consists of can be retrieved. These locations are required for the visualization of the final migration step.
- **Chapter Inference Step:** Establishes the general chapter structure of the target document as the chapters may differ significantly from the source document. The purpose of this step is to establish a hierarchy of chapters and a list of text clusters associated with each of these chapters. The output of this step is a standalone *.json* file which contains a list of all chapters and a list of page-rects which are the text clusters associated with the chapter. This file is required such that sentences of the target document can be established for the sentence matching of the final step.
- **Migration Step:** Automatically tries to match as many sentences that occur in the source document as possible to sentences that occur in the target document. After the user has manually checked and corrected all annotations via the GUI, a new *.apchis* filefor the new version of the brochure can be generated.

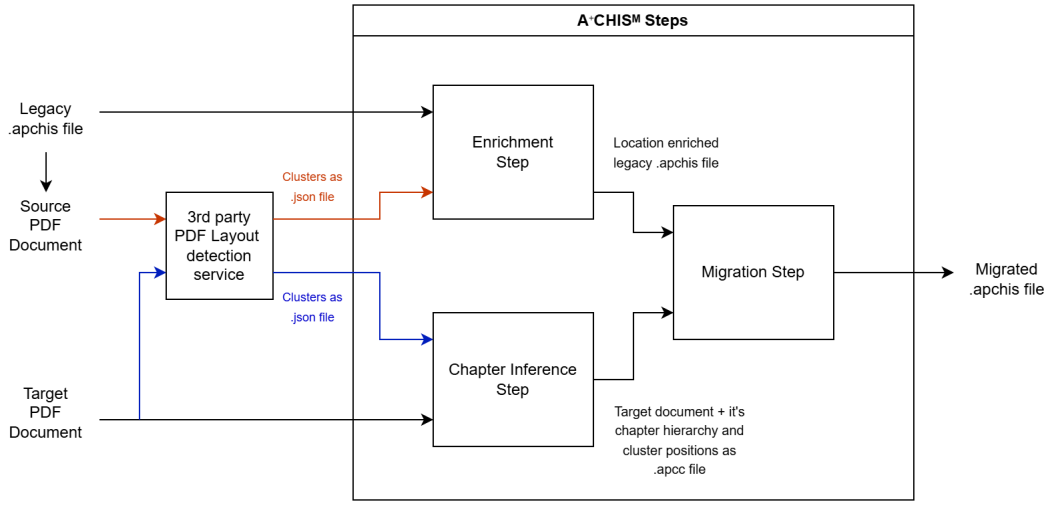


Figure 4.1: High-Level Architecture of the application

4.1 Document Layout Detection Step

When using the default implementation of PDFBox to extract all text content from a page, the algorithm assumes all text to follow a strict top to bottom layouting within the page. Many general purpose documents however, contain instances of small side-information text clusters that may be spread over random locations on pages. An example is shown in Figure 4.2 which leads to multiple problems on whole page text extraction via PDFBox. First, the text in green would be unpredictably mixed with the main text in black on the left. Second, the cluster does not complete the sentence but it continues on one of the clusters on the next page which most of the time is not the first occurring cluster on that page. In this example, in the worst case the page numeration text "9" could be added directly after the hyphen which could possibly break sentence extraction or add additional noise that hinders sentence matching with the other document. Therefore, even if one were able to assume that the first occurring cluster (top to bottom, left to right) continues the sentence, the result would still be error prone. On some general purpose documents it may even happen that multiple grouped clusters interrupt the current content with entirely different topics. Examples like shown in Figure 4.3 where unfinished sentences are interrupted by other chapters or pictures on either the same or the next page before they continue in a text cluster are not uncommon.

Initial attempts to extract coherent text only by grouping text that is either colored consistently or has the same font size did not yield satisfactory results as there were too many exceptions to handle manually. Generally, the text colors and font sizes are not consistent enough to determine which clusters are side information or even which clusters represent section headers of chapters. Initial versions of the application extracted text clusters successfully by using the open source OCR-tool Tesseract[30] and later via distance thresholds but both these approaches were discarded as cluster classification (section-header, listing, normal text etc.) proved to be necessary for the chapter inference step. Therefore, it was decided to use a third party open source PDF layout detection tool[31] to provide input for both the Enrichment step as well as the Chapter Inference step. This tool is provided as a docker container and takes a PDF file as input. It is able to provide both, clusters and the type of text covered by the cluster with sufficient accuracy in JSON format which can be used for further processing. In the background it uses the VGT (Visual Grid Transformer) by Da et al. (2023) [32] which compared to previous document layout analysis models combines both the visual analysis of a page with semantic text analysis.

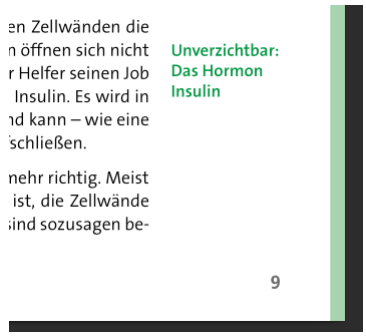


Figure 4.2: Unrelated text clusters next to each other - Source: [12]

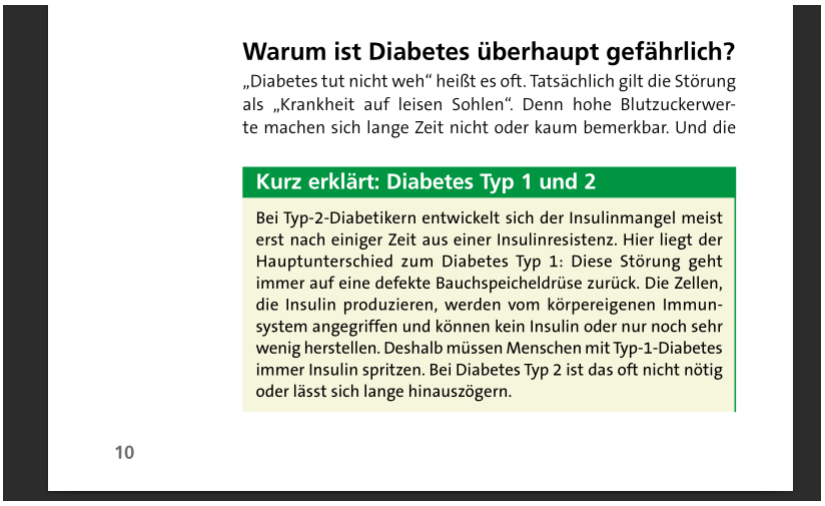


Figure 4.3: Interrupted unfinished sentence from top to bottom - Source: [12]

4.2 Enrichment Step

The Enrichment step is required to provide the initially absent text locations of known chapter sentences for the source *.apchis* file. This step takes as input both the initial *.apchis* fileas well as a *.json* filecontaining the inferred clusters and their type generated by the tool mentioned previously for the source PDF document. In order to match sentences to text in the provided clusters, first it has to be determined in which page range the clusters for each chapters lie. While newer versions of the A+CHIS format contain start page numbers for each chapter, the application uses text stripping on individual pages and then attempts to find the first and last 3 sentences within the chapter which is a cheap operation performance wise. If at least one of those sentences is found for each the first and last ones, then the preliminary start and end page number has been determined. The approach for automatic inference of text locations for the known sentences within the provided clusters of a cluster happens based on a priority system as shown in Figure 4.4 for performance reasons. In each of the approaches outlined, a match is first attempted via Levenshtein-Distance. Only if the calculated Levenshtein-Distance of a potential match is high enough but not quite sufficient, the BERT embeddings are calculated and the cosine-similarity is compared against a threshold for a semantic match.

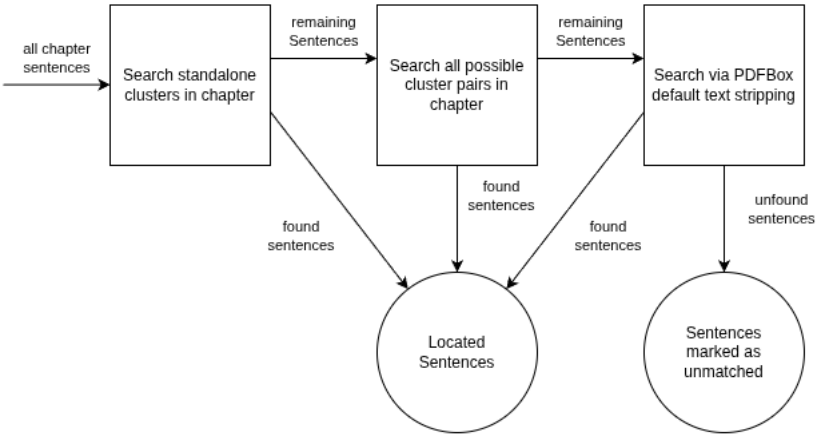


Figure 4.4: Flow for automatic location inference during Enrichment step

After the automatic location inference process has concluded, the user can review the findings and make manual adjustments. The user is presented with an Enrichment view in the GUI which can be seen in Figure 4.5. When clicking any sentence with any known text locations in the content browser on the left of the view, the shown *.pdf* file will scroll to this position such that the user can view the inference and make their adjustments if necessary, without searching. Textlocations can both, be added and removed via selection marquee as shown in Figure 4.6. Once the user is done with their adjustments, the user can finish by saving an *.apchis* file that contains everything the input *.apchis* file contained as well as the already mentioned *.json* file named *text_locations.json* which contains the just determined text locations. The user can re-open an *.apchis* file that already contains such a file and the automatic inference step will be completely skipped. This allows for later edits in case mistakes are discovered. Once a document has been migrated to a newer version, all subsequent versions of the *.apchis* file should always contain this file. Therefore, an enrichment step is theoretically only required to be done once during the entire life cycle of a document used in A+CHIS.

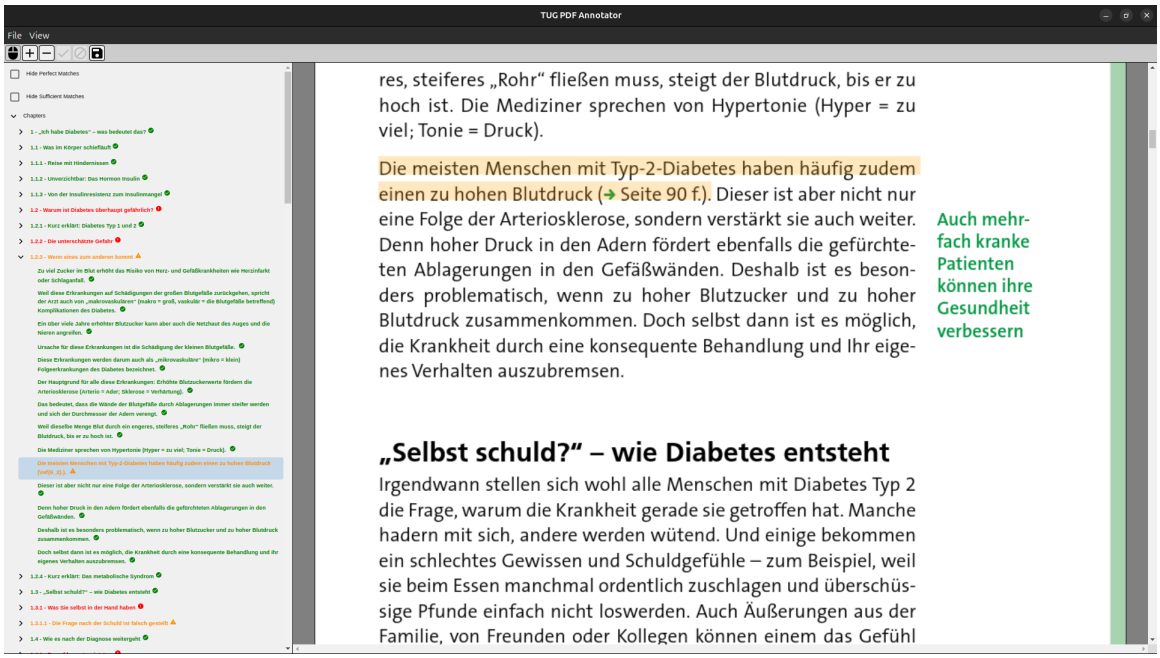


Figure 4.5: Highlighted inferred sentence locations in Enrichment view

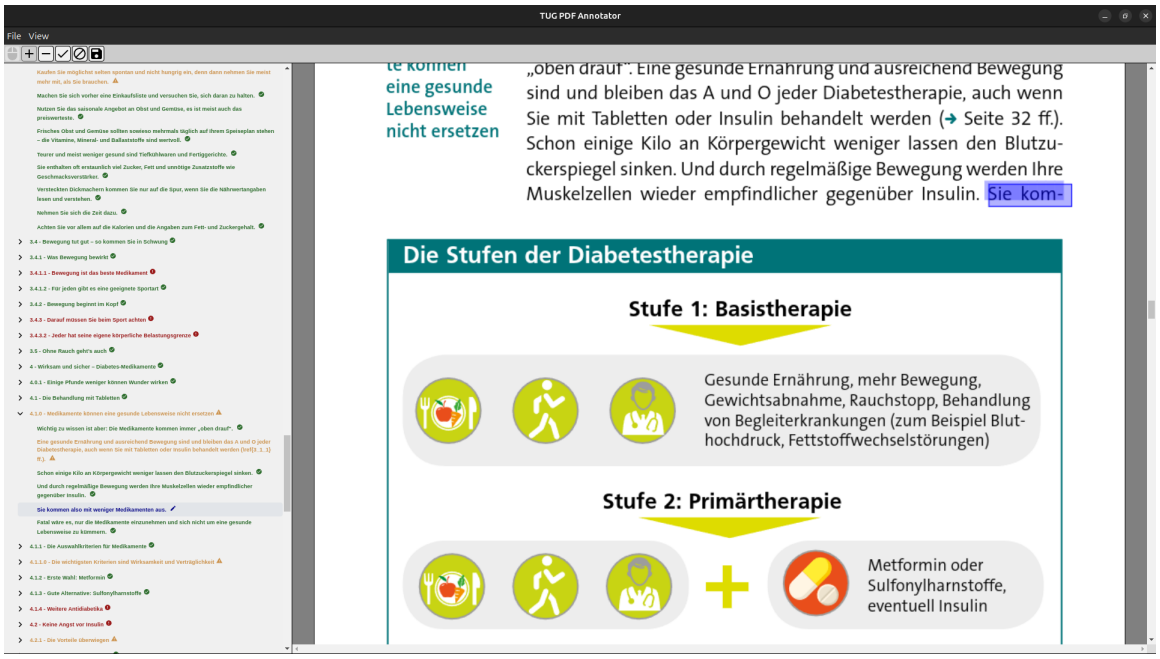


Figure 4.6: Adding missing text locations to sentence in Enrichment View via selection

4.3 Chapter Inference Step

The Chapter Inference step is required to establish the chapter structure of the target document. It also associates all text clusters that occur within the document to a chapter and provides the user the ability to manually connect clusters that belong together. An overview of the processing flow of this step can be seen in Figure 4.7. The automatic inference of the chapter hierarchy tries to first find all section headers within the provided text clusters. It then sorts them according to position from top to bottom and then establishes which font sizes are likely to be representing different sub chapter levels. Beneath a section header, as long as other section headers occur that have smaller font sizes, additional sub chapter levels are created within the current chapter hierarchy. Whenever a large font size occurs for a following section header, it is determined by font size which other chapter this new chapter will be a sibling off. The provided clusters from the third party service are not completely accurate. While most inferred clusters are typed correctly, many adjustments still have to be made to ensure correctness of both the chapter hierarchy and clusters.

The pre-provided text clusters by the third party layout detection tool have to be pre-filtered to not have negative dimensions or be located outside of the visible space of a page in order to be considered. Not doing so lead to corrupt clusters which break the graphical user interface which would draw clusters over entire pages in the PDF viewer that were not interactible.

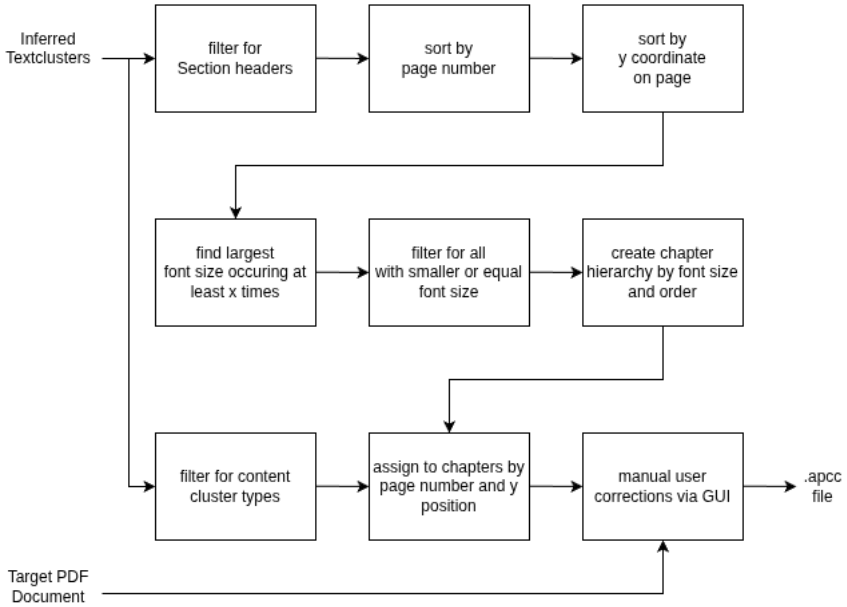


Figure 4.7: Process flow of Chapter Inference step

For manual corrections the user has multiple tools at their disposal as shown in Figure 4.8. Inside the content browser on the left side, the user is presented with the current presumed hierarchy of the target document. A click on each of the chapters will scroll the PDF view to the position of the section header that is associated with the clicked chapter. Inside the PDF viewer, all content clusters are highlighted in different colors according to their type. Most content clusters are either considered pure text (green) or listings items (yellow). Visualized text clusters that do not belong to the currently selected chapter are displayed with a stronger alpha channel. The type of each cluster can be changed by clicking on it as a modal will open prompting the user to select the correct type. The user has the ability to both create new clusters by selection marquee and delete existing ones. Whenever a text cluster is changed into a section header or a new text cluster is created as a section header, a new chapter will be inserted at that position. Vice versa, deleting a section header or changing its type will cause the corresponding chapter to be removed from the hierarchy. There are buttons to both promote and demote the currently selected chapter within the hierarchy. A chapter can only be demoted if it has a direct sibling that occurs before it. It will then become a sub chapter of that sibling. Promoting a chapter will cause it to become a sibling of its current parent chapter. Additionally within its current hierarchy level, the selected chapter can be moved up and down to change its position in respect to its siblings. As shown in Figure 4.9 content text clusters

that belong to each other have to be linked manually by the user. A link chain between clusters can be arbitrary long and is visualized via red arrows in the PDF viewer. These links ensure that sentences can be extracted efficiently and correctly when they span over multiple clusters. Once the user is done with their manual adjustments the save button can be used to zip the hierarchy and cluster information as well as the target document into an *.apcc* file that can be used as an input for the final Migration step. As the *.apchis* file of the source document is not an input for this step, we need this separate file type to store the results of this step. An existing *.apcc* file can be re-opened for Chapter Inference to make further corrections if mistakes are discovered later.

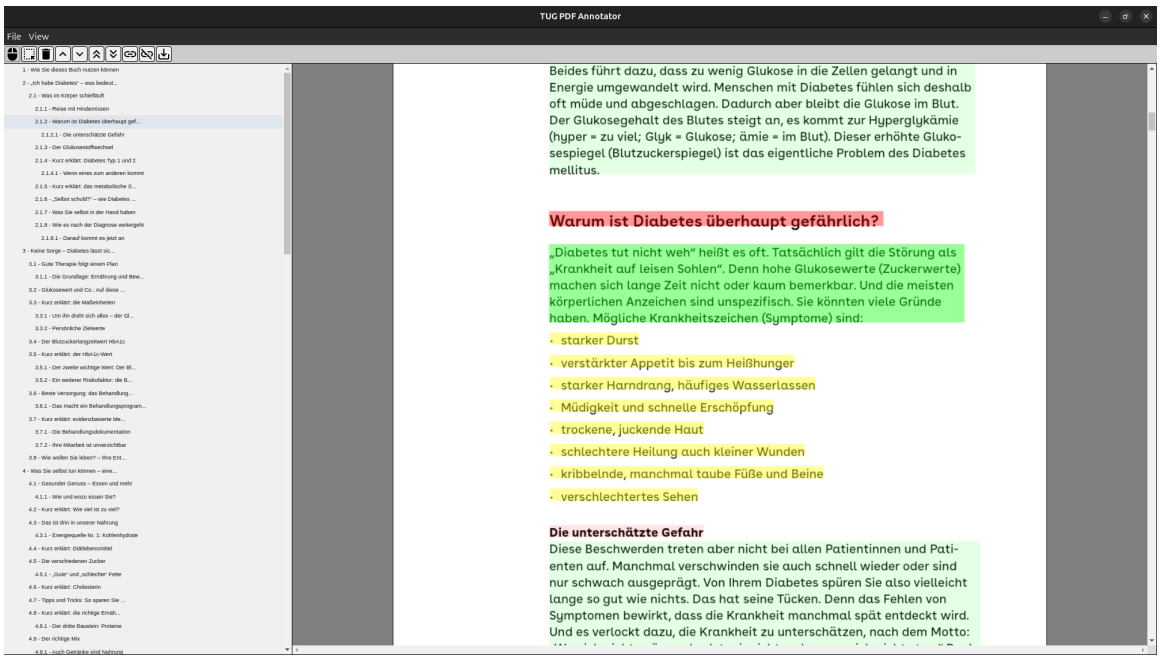


Figure 4.8: Chapter-Inference-View

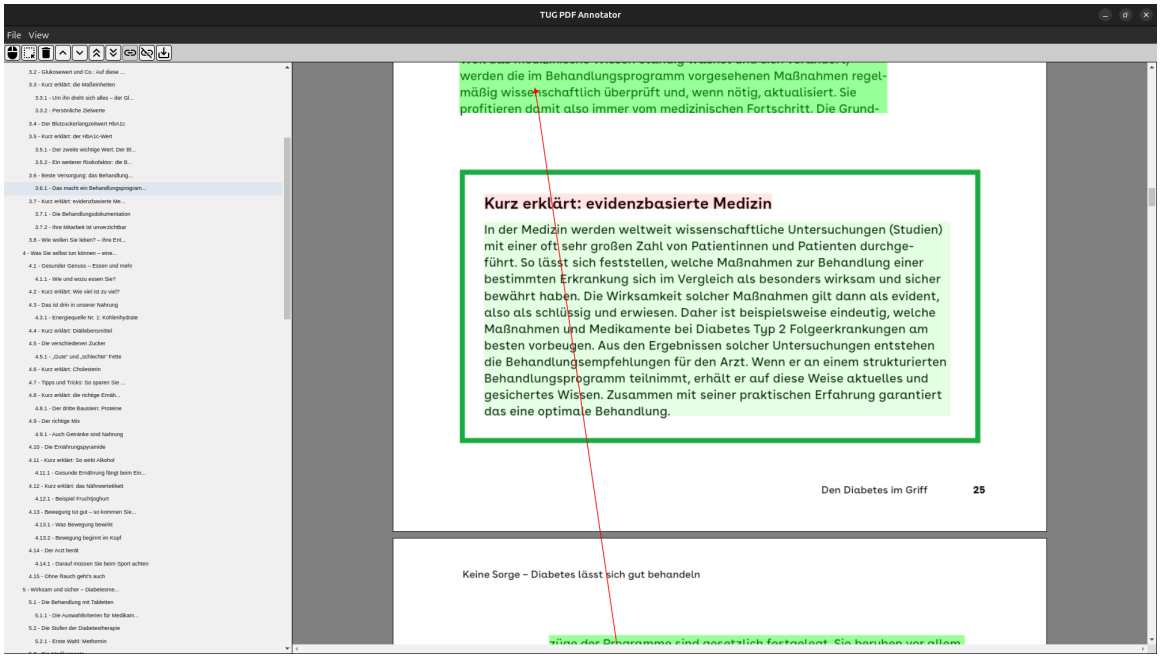


Figure 4.9: Linking two content clusters

4.4 Migration Step

The migration step takes the output files of both previous steps as input. This step attempts to find as many sentences that occurred in the source *.apchis* filein the target document automatically. First the inference process tries to strip all non-alphabetic characters from compared sentences and then attempts an absolute match between the candidates. For sentences that cannot be matched absolutely, a match via Levenshtein-Distance is attempted with all non-alphabetic characters still stripped from both candidates. For all remaining unmatched sentences, a match via cosine-similarity of the BERT embeddings is attempted for candidates that have a sufficient Leventhstein-Distance match. All remaining unmatched sentences are assumed to require manual intervention to find or discard.

As the processing steps outlined above are brute force comparisons, some performance optimizations were made to shorten the time the user has to wait. Generally, the computations and comparisons in this step are heavily parallelizable as they only loosely depend on each other. Therefore, the entire inference logic for this step is abstracted into a processing unit which holds multiple key-value maps in memory that are able to handle concurrent accesses from different threads. As the very first comparison level is a complete character matching as mentioned above, this disqualifies sentence candidates with multiple occurrences in the same document as those are theoretically possible for short sentences. Therefore, a set of such sentences is precomputed for both the source and the target document contexts. As the computation of these sets already requires the filtered alphabetic characters only of each sentence, these translations can be stored in maps as they will be required in the first comparison step as well. Additionally, a quick mapping from sentences to their respective chapters is constructed which is used by the processing levels. The absolute character matching level establishes a mapping between chapters of the source document and chapters of the target document as soon as a single match is found. All subsequent levels like matching by Levenshtein-distance or BERT embeddings require such a source to target chapter mapping to be present or otherwise the contents of these chapter have to be skipped as the amount of possible comparisons would lead to a very long execution time.

After automatic inference, the user is presented with a side to side comparison view as shown in Figure 4.10 where both the source and target PDF document can be compared against each other. The browser on the bottom shows all matched, incomplete and unmatched annotations. A click on one of these sentences in the browser will scroll both shown PDF Viewers to the location of the match which is then highlighted on both sides. The edit mode in this view allows for both the adding and removing of sentences from the annotation matching. When in add-sentence mode, each sentence that is not yet added to any annotation within the same chapter as the sentences that are already added, will be highlighted when hovered and can be clicked to add it. In Figure 4.11 the missing sentence is highlighted

for addition on hover. Vice-versa, when in remove mode, sentences that are already part of the annotation are highlighted when hovered and a click removes them from the annotation. The tag of the annotation can be adjusted as well with a built in search by name and id.

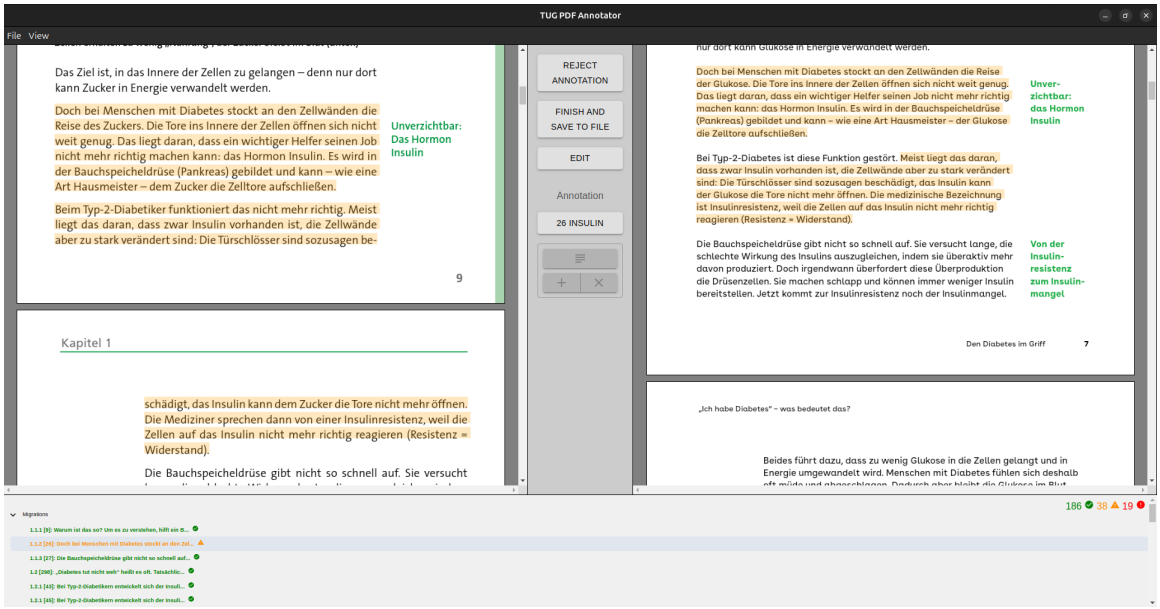


Figure 4.10: Migration View showing an incompletely matched sentence

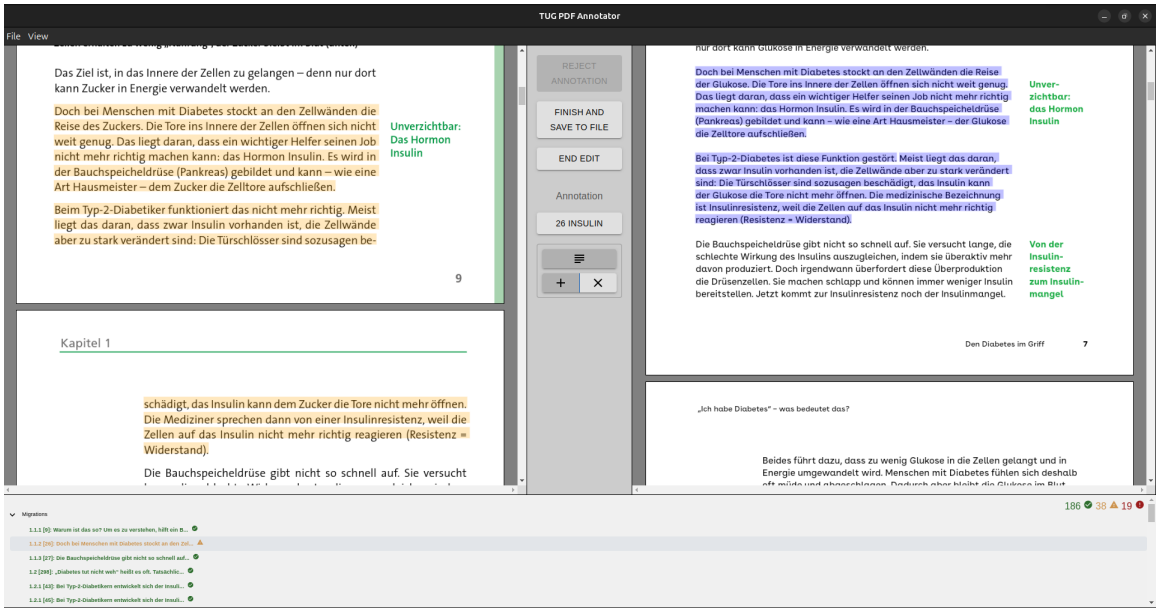


Figure 4.11: Auto highlighting the hovered un-added sentence

5 Implementation

Many different technologies are combined to develop A^{+}CHIS^{M}. Table 5.1 shows an overview of the most important technologies used. Generally the application code is divided into a back-end and front-end project where the former is a Java Spring project and the latter is an Electron desktop application written in React and Typescript. The decision to use Java in combination with the Spring framework for the back-end was made based on the following benefits:

- Simple to implement multithreading capabilities,
- Popular PDF manipulation and extraction library: PDFBox[9],
- The author's familiarity of working with these technologies.

The decision to pick React as the front-end technology was mainly made for the flexibility benefits a modern web-stack provides for GUI development in combination with pdf.js[33] while still having access to the local file system due to being wrapped in an Electron window. The communication between the back-end and front-end applications happens via websockets which is sufficiently performant for A^{+}CHIS^{M} to pass the content of the entire PDF documents without noticeable delay. Communication via shared memory, while in theory arguably the fastest way of communication between both processes is difficult to implement in an electron project as native APIs are nonexistent while named pipes would require a platform dependent implementation.

Technology	Usage
Java & Spring Framework[34]	The back-end application stack. Communicates via websocket messages with the GUI, reads and splits the input files, processes automatic inference logic and writes the output files.
React[35] & Electron[36]	The front-end application stack (GUI). A desktop application built with a web based framework that communicates with the back-end via websocket messages. The content of a web browser can be displayed inside a sand-boxed environment with limited system access that can communicate with the more privileged part of the application via narrow defined APIs.
Gradle[37]	The overarching build tool for the entire project. Primarily manages dependencies of the back-end stack, but also partially generates front-end interfaces of messages defined by back-end code and indirectly invokes build instructions for the front-end code base.
Apache PDFBox[9]	Library for processing and modifying PDF documents. Primarily used by A$^+$CHIS^M to extract all text tokens within pre-defined rectangular areas and aggregating document characteristics like common font sizes.
Mozilla PDF.js[33]	front-end library for parsing and rendering PDF content into the GUI. Primarily used by A$^+$CHIS^M to render the document pages content onto canvases and to determine the position of individual text tokens for marquee hover highlighting.
Apache Commons Text[38]	Library providing implementations for string based algorithms. Used by A$^+$CHIS^M for its Levenshtein-Distance implementation.
Apache OpenNLP Sentence Detector Model[39] & OpenNLP-Tools[40]	Model and API to split a provided text chunk into individual sentences.
Model: Multi cased BERT model[41] & easy-bert[29]	The used BERT model and the library to interact with it from within the back-end Java code base. The provided API takes a sentence as input and provides the embedding vector as output.

Table 5.1: Main technologies & libraries used

5.1 Visualization Principles

All GUI views the user can interact with in A$^+$CHIS^M follow the same principle. The majority of the screen space is occupied by a PDF viewer in which the actual PDF document being edited is rendered. On a smaller fraction of the screen space the user is presented with a content browser by which they can quickly navigate through content that is relevant for the current task they are performing. Interaction with this browser, scrolls the interacted with content into view on the PDF viewer to enable efficient edits of content that could not be inferred automatically.

The content shown in each view's browser, uses a color scheme which is consistent within A$^+$CHIS^M. Green colors indicate a successful automatic inference of something or a done manual edit while an orange color indicates a partial automatic match that has not been corrected manually yet. Red color indicates something where not even parts of it could be inferred automatically and blue color indicates that something is currently being edited.

Two of the three views enable the user to add or remove content inside the document they are editing by selection marquee. This is inspired by the PAWLS[42] tool which is shown in Figure 5.1. In PAWLS a selection marquee makes words that are either covered or partially overlapped by the marquee rectangle, snap visually. In the case of A$^+$CHIS^M, a selection rectangle visualizes individual characters of which the PDF based bounding box is overlapped by the selection rectangle. This is only a visual indication however, to signal to the user that the selection actually affects the text within the PDF viewer. When the user does draw a selection marquee over text, it is not the GUI that filters the characters that are covered by the rectangle. Instead, when a selection is confirmed by the user by releasing their mouse button, this information is sent to the back-end of A$^+$CHIS^M and based on the current editing context, the back-end

extracts all characters within that area from the document via PDF box and performs an action with them.

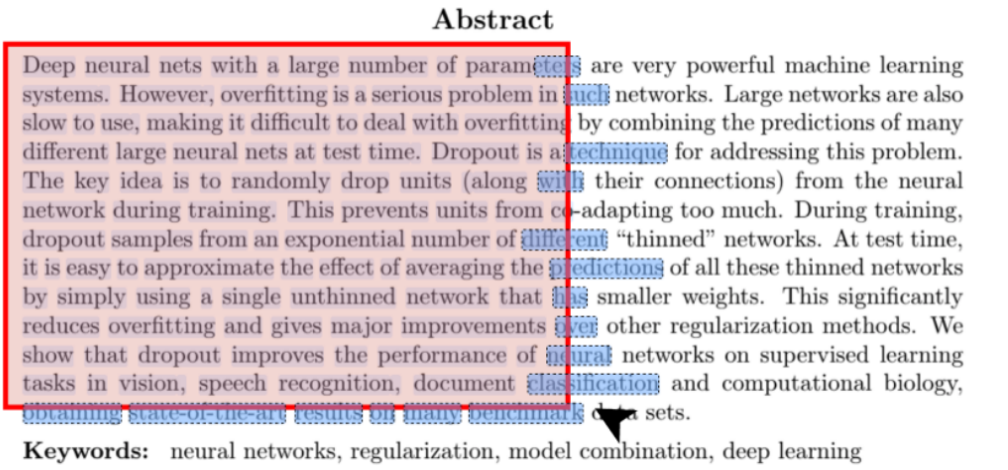


Figure 5.1: Token snapping in PAWLS marquee selection - Source: [42]

5.2 PDF Viewer Implementation Design

The most popular open source option for displaying PDF content in a web technology based application is pdf.js [33] which is distributed as both a bare-bones library to render PDF content onto canvases or as a ready to customize viewer that is also used by the Firefox web browser. Although there also exist some other commercial and freemium options, most free/open-source wrappers for different front-end frameworks like react-pdf[43] use pdf.js under the hood. We decided to integrate the bare-bones pdf.js library into A$^+$CHIS^M instead of using the prebuilt viewer as the heavy customizations that were required for this project were considered harder to do otherwise.

The PDF content of each individual page is rendered onto a HTML canvas element. The implemented PDF viewer, therefore is a container element that stacks many canvas elements vertically with fixed distance offsets. Each of these canvases only renders its PDF page content when it comes into the users view, which is necessary for performance of A$^+$CHIS^M for large documents like the brochure. For all the different GUI views of their respective step (**Enrichment Step**, **Chapter Inference Step** and **Migration Step**) there exist different tools inside the viewer for manual interactions that require custom visualizations. Those custom visualizations are achieved by having multiple layers of transparent canvas elements stacked directly on top of each other for every page. This is necessary because a single canvas does not have multiple layers on which different things can be drawn and cleared without affecting other content. Meaning, if a single canvas were to render both the PDF content of a page as well as text highlights for an annotation, clearing that highlight would require the entire PDF content to be re-rendered. This would pose a performance challenge and is also not trivial to implement. For each of the different views A$^+$CHIS^M provides, the PDF page container always provides the same order of hierarchy for its layers. The first layer, as described above merely renders the PDF content of a page. The second layer draws all text highlights on a page where an individual highlights consists of a list of rectangles and color in which the highlight is to be shown with a fixed alpha channel. Each provided highlight can also provide custom handlers for hovering and un-hovering the highlight and for user mouse clicks. The highest layer is called the Selection layer and provides the visualization of the marquee selection rectangle that the user can draw over a canvas by clicking and dragging over the content of a page. The visualized rectangle is a rectangle with alpha channel and a border without an alpha channel.

The chapter-inference view requires visualizations to indicate links between different pages which will be elaborated on later. These visualizations are achieved by another invisible canvas that spans over the container that contains all of the vertically stacked canvases. In addition, there is also a separate tooltip layer that spans over the entire content of A$^+$CHIS^M's GUI which enables the display of tooltips

with interactive content for when a user performs manual actions like for example hovering or clicking highlights in a view. An example for such a tooltip can be seen in Figure 5.2.

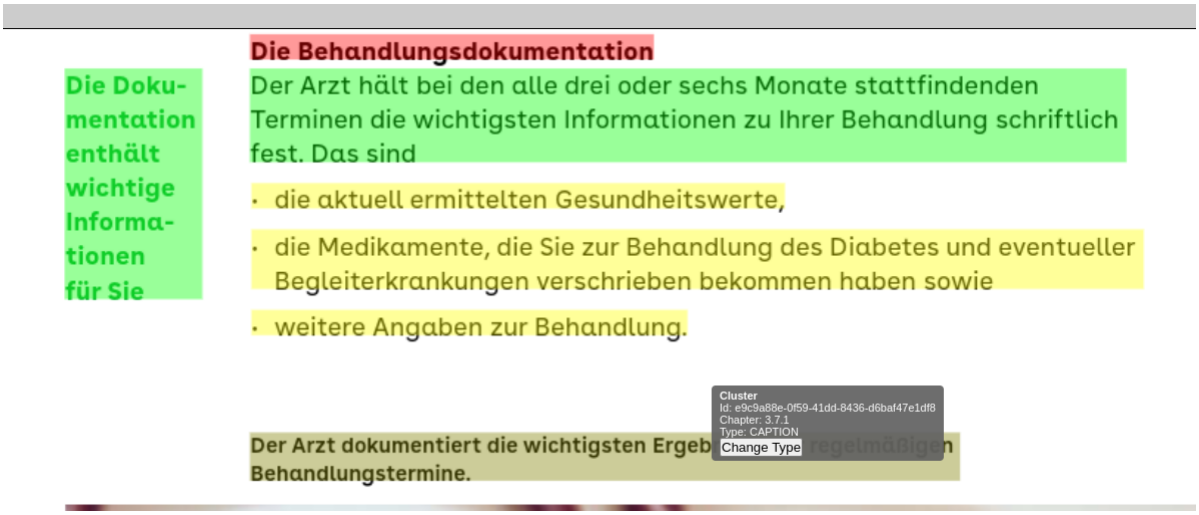


Figure 5.2: Tooltip shown for a highlight clicked by the user in the Chapter-Inference view

5.3 Application Messaging

As mentioned before, the front-end and back-end parts of A^{+}CHIS^{M} communicate with each other via websockets. In order for the Spring framework to map websocket messages correctly, the connection needs to upgrade to the STOMP protocol. The stomp protocol turns the websocket connection which is only a bi-directional pipe into a protocol following a publish subscribe pattern with acknowledgements that messages were received.

5.4 Text-Token Information Extraction by Area

Apache PDFBox[9] provides an API by which text from a page or given rectangular area within a page can be extracted. These APIs are called `PDFTextStripper` and `PDFTextStripperByArea` respectively. However, by default this API only provides the extracted text in plain text as output and not as a list of individual tokens and more detailed information like their exact position, font and coloring. As the API does use a `TextPosition` object representation for its processing in the background, both APIs were customized to gather and expose these lists of filtered `TextPosition` objects for further use in A^{+}CHIS^{M}. While the necessary modifications necessary to extract the `TextPosition` objects themselves were straight-forward, the extraction of text color required internal customized implementations of how PDFBox reacts to the PDF content stream operators `SetNonStrokingColorN(scn)`, `Save(q)` and `Restore(Q)`. This was required to store the current stroking color state and assign it to all text positions extracted within the time until another stroking state was pushed or a previous one was restored.

6 Use Case Example

In this chapter we will discuss how well A$^+$CHIS^M performs on the actual use case of the brochure on T2DM by the AOK [12] which is the primary CHIS currently being tested with the A$^+$CHIS project.

Both the source and target document of the brochure contain so called "Kurz-Erklärt" - Chapters (KE-Chapters). These are mini-chapters which are essentially just boxes with a headline and some additional information inside. Examples like shown in Figure 4.3 where unfinished sentences are interrupted by KE-Chapters or pictures on either the same or the next page before they continue in a text cluster are common within all versions of the brochure.

6.1 Results

The runtime durations of the automatic inference processes for the different outlined application steps are very dependent on the hardware on which the application is tested as well as the amount of pages for the CHIS being migrated. All mentioned runtime performance measurements are based on a state-of-the-art consumer hardware. All results and measured computation durations refer to the T2DM brochure by the AOK [12] using the hardware specified in Table 6.1. It should be noticed however, that the performance especially scales with CPU performance and amount of possible simultaneous thread executions. Additional RAM allows to allocate more independent models for sentence extraction and calculating BERT embeddings that can be simultaneously used by different threads which also speeds up the process to some extend. All duration measurements of manual work refer to us using A$^+$CHIS^M.

Table 6.1: This table shows some performance relevant system specifics for the system A$^+$CHIS^M was tested on.

Component	Type used
CPU	AMD Ryzen 9900X
RAM	64GB @6000MHz
OS	Ubuntu 24.04 LTE

The thresholds used to determine whether or not two compared pieces of content or sentences match also heavily influences the runtime duration of automatic inference tasks. Lower thresholds lead to quicker run-times but more false or incomplete positives.

Additionally, a cognitive walkthrough of A$^+$CHIS^M was performed together with stakeholders of the A$^+$CHIS project. Our application was well received.

6.1.1 Results - Enrichment step

The enrichment step's primary purpose is to prepare the source *.apchis* file for usage in the migration view. Therefore, an exact match is not mandatory as long as the visualization of the matched content is sufficient to make a comparison with the target document possible in the view. The parameters shown in Table 6.2 were chosen to perform the automatic text location inference on the source document. We deemed them to strike a good balance between runtime duration and correctness for this step. Table 6.3 shows the results achieved with those parameters.

Table 6.2: This table shows the configuration parameters used and the runtime achieved.

Configuration	Value
Levenshtein match threshold	0.95
BERT match threshold	0.85
Bert fallback tolerance	0.10
Runtime	19.91 s

Table 6.3: This table shows the achieved results with the parameters outlined above.

Property	Number
Total sentences to match	1158
Sentences matched total	1122
Perfect sentence matches	1062
Imperfect sentence matches	60
Sentences unmatched total	36
Sentences matched by BERT	22
False positive matches	0

A perfect match in this context means that a sentence has an exact character match with the compared one, given that all white spaces and hyphens were stripped from both. Given the high required thresholds to achieve a match in this test, the imperfect matches can still be considered sufficient to visualize those matches on the source document during the migration step. The amount of imperfect matches listed aggregates all matches that are not perfect matches. Manual correction of the imperfect matches and unmatched sentences within $A^{+}CHIS^{M}$ in this test scenario took us fifteen minutes.

The imperfect matches in this test were caused by the following reasons:

IM1 When a specific page is referenced in the source brochure document, the pre-stored sentences were manually adjusted to reference the chapter instead with different syntax.

IM2 Characters representing bullet points in the brochure source document were manually replaced with the character sequence "[-]" in the pre-stored sentences.

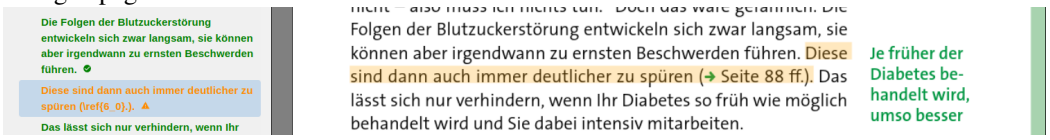
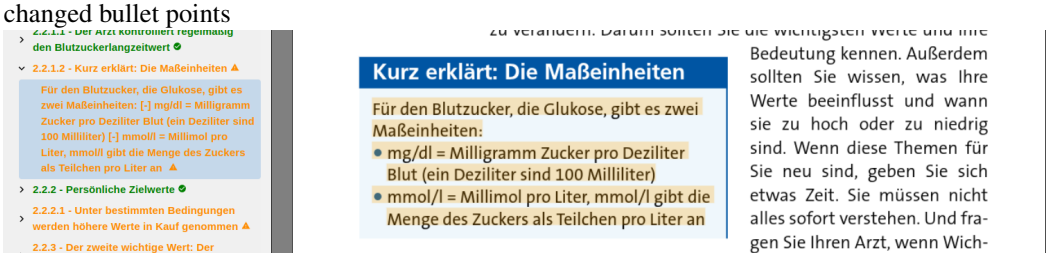
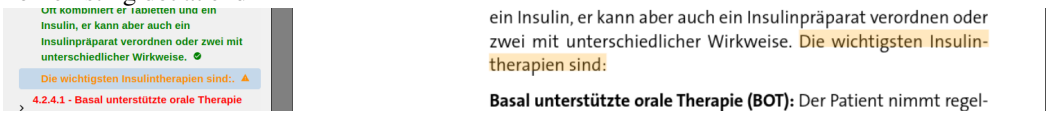
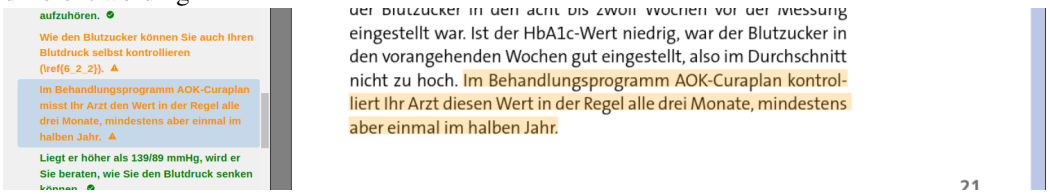
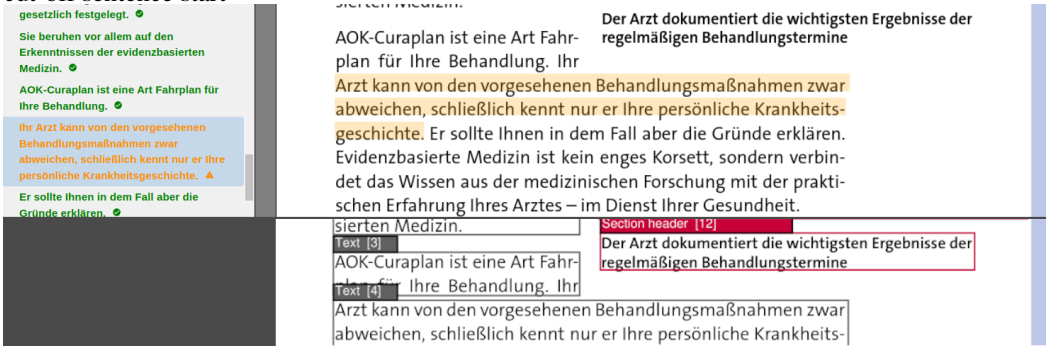
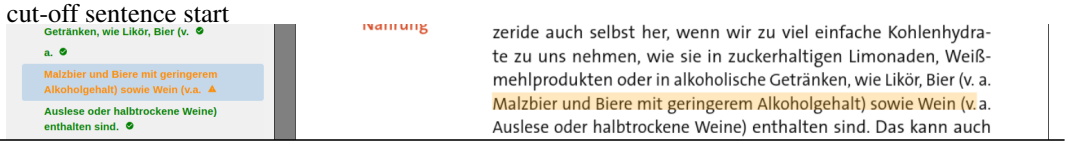
IM3 The pre-stored sentences end with punctuation but there is no punctuation in the real document.

IM4 The wording of pre-stored sentences differ from the source document.

IM5 The first or last few characters of the sentence are located in another cluster but the partial text matches sufficiently to surpass the configured thresholds.

IM6 The pre-stored content is not an actual sentence and the sentence matching logic from the application differs.

Table 6.4: Examples for imperfect matches

	Example
IM1	changed page reference 
IM2	changed bullet points 
IM3	non-existing dot at end 
IM4	different wording 
IM5	cut-off sentence start 
IM6	cut-off sentence start 

Unmatched content could not be matched because of the following reasons:

UM1 All the reasons that also lead to imperfect matches but to a severity where configured matching thresholds cannot be met.

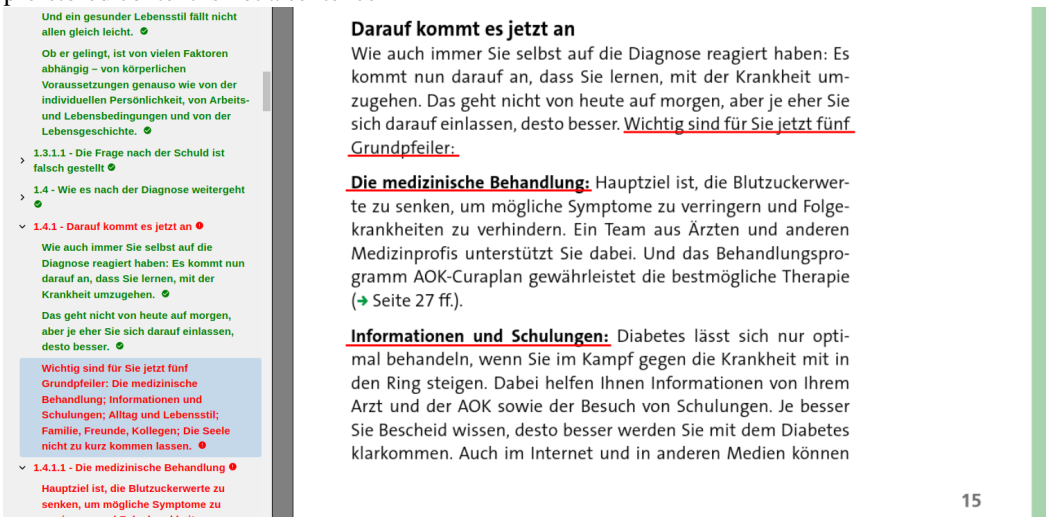
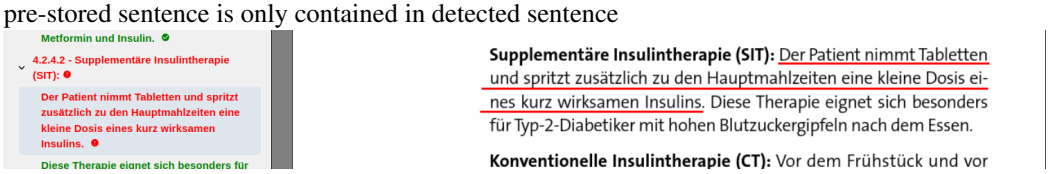
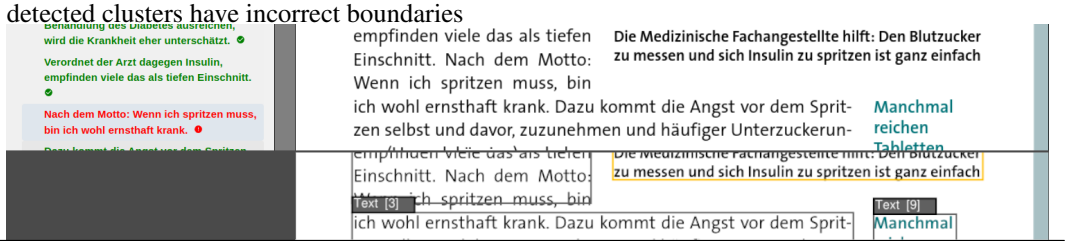
UM2 There is more than one entire page between two clusters that a sentence belongs to. This will not be matched as the inference algorithm only considers the following page to search for a pair cluster.

UM3 The pre-stored content violates the rule that stored contents are extracted sentences and instead parts of multiple sentences are referenced as one piece.

UM4 The pre-stored sentence does not reflect the entirety of the sentence as the sentence model used by the application would extract it.

UM5 The clusters detected by the employed third party tool do not line up sufficiently with the content that contains the sentence.

Table 6.6: Examples for unmatched content

	Example
UM3	pre-stored content is not a sentence 
UM4	pre-stored sentence is only contained in detected sentence 
UM5	detected clusters have incorrect boundaries 

No document specific adaptations were made to A$^+$CHIS^M to deal with these problems for the sake of keeping the solution general. Given that this test attempt yielded a result that made manual corrections possible in a reasonable amount of time it did not seem beneficial to sacrifice compatibility with possible future versions of the same document for a few additional minutes gained by matching the remaining content of the initial document correctly.

6.1.2 Results - Chapter Inference Step

The majority of all chapters and clusters could successfully be inferred by using the target document and clusters produced by the externally used layout detection service. Compared to the enrichment step, this step did not require any significant processing time exceeding a few seconds. As the provided clusters do contain some mistakes however, there were some corrections that had to be made. The linking of connected content clusters had to be done manually as it is not part of the automatic inference process. Both, fixing the automatically inferred mistakes as well as linking connected clusters took around one hour of focused work for us to fix within A$^+$CHIS^M. The following kind of mistakes were discovered during manual correction:

CIP1 Text not being captured at all by clusters.

CIP2 Non-detected section headers that are covered by normal text clusters.

CIP3 Detected ghost clusters which capture invisible, irrelevant text.

CIP4 Singular clusters spanning over entire tables, covering content that is not fit for extracting sentences.

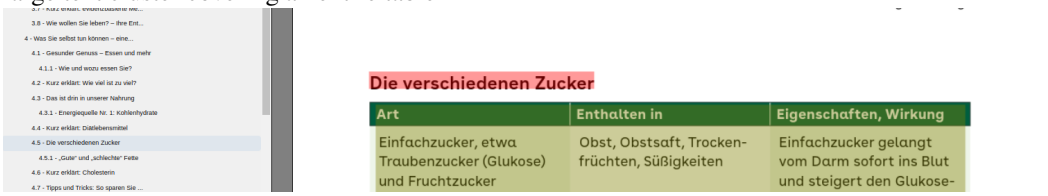
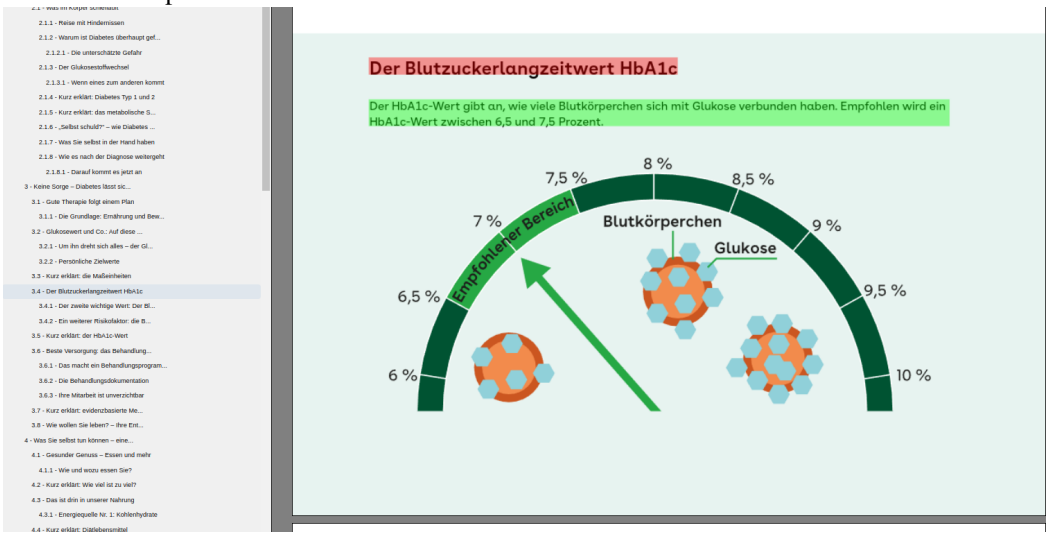
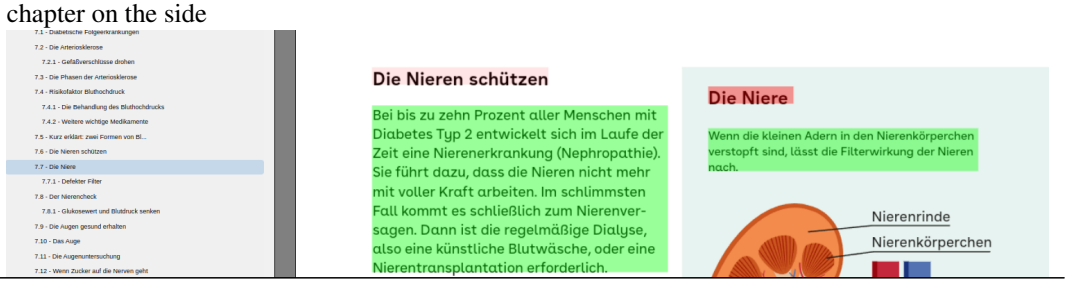
CIP5 Some chapters are structured like KE-Chapters but they do not have the common "Kurz Erklärt" prefix and therefore cannot be recognized as such.

CIP6 There are a total of twelve chapters in the target document that interrupt the content of other chapters which do not contain much text but often contain a large figure. These chapters are visually indicated by a slightly blue colored background. This causes clusters that occur after these in-between chapters to be assigned to those chapters instead of to the interrupted ones. Two of these chapters break the usual chapter hierarchy completely as they split the content of a page by left and right. There are another seven such in-between chapters with a green background that give some side information but contain no figures. As there is no common pattern by which to distinguish such chapters (like with the KE-Chapters) easily that is consistent between different document versions, manual corrections are required.

CIP7 Similar to chapters containing mostly large figures, there are seven in-between chapters, giving information about a specific side topic without a section header with common prefix, which need to be handled manually.

CIP8 Some sentences with bold fonts are interpreted as section headers even though they are just normal texts belonging to a chapter.

Table 6.8: Examples for mistakes in chapter inference

	Example
CIP3	invisible content is captured 
CIP4	large text cluster covering an entire table  **Die verschiedenen Zucker** Art / Enthalten in / Eigenschaften, Wirkung Einfachzucker, etwa Traubenzucker (Glukose) und Fruchtzucker / Obst, Obstsaft, Trockenfrüchten, Süßigkeiten / Einfachzucker gelangt vom Darm sofort ins Blut und steigert den Glukose-
CIP6	in-between chapter  **Der Blutzuckerlangzeitwert HbA1c** Der HbA1c-Wert gibt an, wie viele Blutkörperchen sich mit Glukose verbunden haben. Empfohlen wird ein HbA1c-Wert zwischen 6,5 und 7,5 Prozent. 6 % 6,5 % 7 % 7,5 % 8 % 8,5 % 9 % 9,5 % 10 % Empfohlener Bereich Blutkörperchen Glukose
CIP6	chapter on the side  **Die Nieren schützen** Bei bis zu zehn Prozent aller Menschen mit Diabetes Typ 2 entwickelt sich im Laufe der Zeit eine Nierenerkrankung (Nephropathie). Sie führt dazu, dass die Nieren nicht mehr mit voller Kraft arbeiten. Im schlimmsten Fall kommt es schließlich zum Nierenversagen. Dann ist die regelmäßige Dialyse, also eine künstliche Blutwäsche, oder eine Nierentransplantation erforderlich. **Die Niere** Wenn die kleinen Adern in den Nierenkörperchen verstopft sind, lässt die Filterwirkung der Nieren nach. Nierenrinde Nierenkörperchen

6.1.3 Results - Migration Step

For testing the migration step, the outputs of the test runs for both previous steps were taken as input. After automatic migration inference, an individual chapter annotation consisting of one or more sentences was considered *complete* if the amount of matched sentences was equal to the amount of source sentences. If some but not all sentences of an annotation were matched, the annotation was considered *incomplete* and if no sentence at all could be matched, the annotation was considered *unmatched*. The automatic migration was performed with configuration settings shown in Table 6.10 and yielded the results shown in Table 6.11. A manual correction of all annotations was performed by us within fifteen minutes using the GUI provided by $A^{+}CHIS^{M}$.

Table 6.10: This table shows the configuration parameters used and the runtime achieved

Configuration	Value
Levenshtein match threshold	0.75
BERT match threshold	0.98
Levenshtein threshold for Bert fallback	0.30
Runtime	19.24 s

Table 6.11: This table shows the results achieved during automatic content matching of the Migration Step.

Total source chapters	141
Total target chapters	149
Total source sentences	1158
Total target sentences	1731
Total annotated source sentences	1122
Total sentences of completely matched	806
Total sentences of incompletely matched	217
Total sentences matched	1023
Total sentences unmatched for incomplete match	54
Total sentences unmatched for completely unmatched	45
Total sentences unmatched	99
Total chapter annotations matched	187
Total chapter annotations incompletely matched	37
Total chapter annotations unmatched	19

The following different cases of unmatched content were observed:

MP1 All content of the annotation is too different from the original and nothing could be matched.

MP2 All content could be matched but a single sentence in between was not meeting required thresholds.

- While a matching for such in-between sentences would be trivial based on the surrounding matching context, the decision was made to make the user deal with all unmatched cases manually.

MP3 The original content violates the rule of being only based on full sentences but instead spreads its content over individually spread text pieces.

MP4 Sentence quite similar semantically but not sufficiently equal for a BERT match.

MP5 The text content is very similar but no match could be achieved because the target version contains abbreviations for multiple words instead of spelling them out.

MP6 The content was matched correctly but the source is split into more pseudo-sentences.

MP7 The text considered to be one sentence in the source is only a part of a bigger text unit considered to be one sentence in the target document.

- Such cases have to be fixed manually in the resulting *.apchis* file. A reassignment of correct text locations can be achieved by using the new document as input for the enrichment step. As text locations are already present, no automatic inference will take place and the user can modify text locations manually.

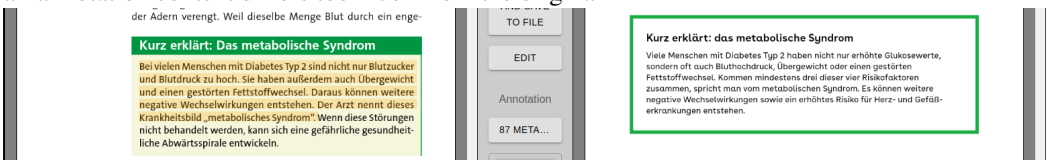
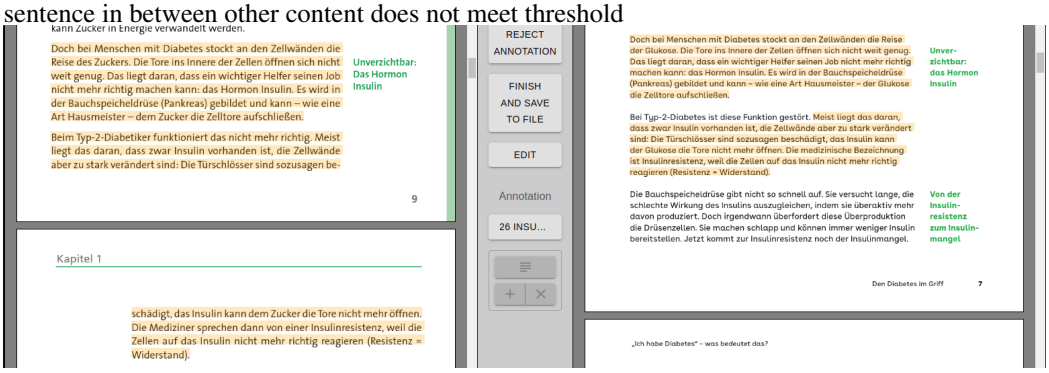
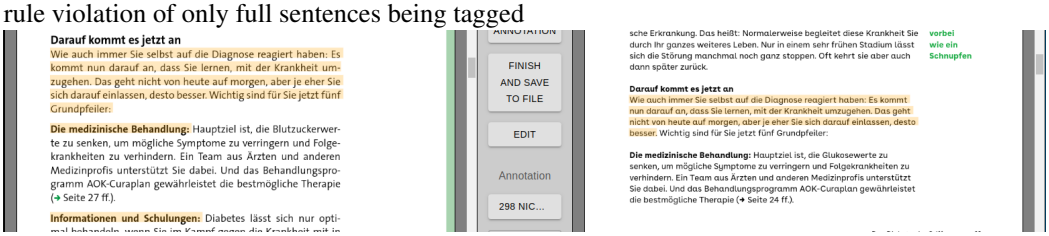
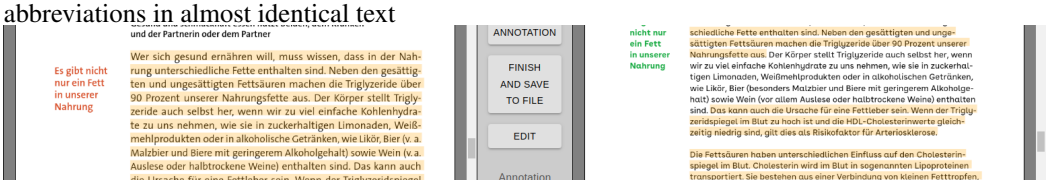
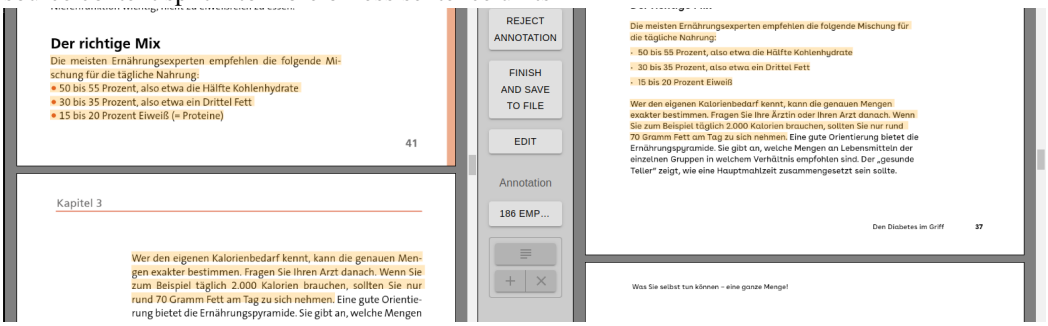
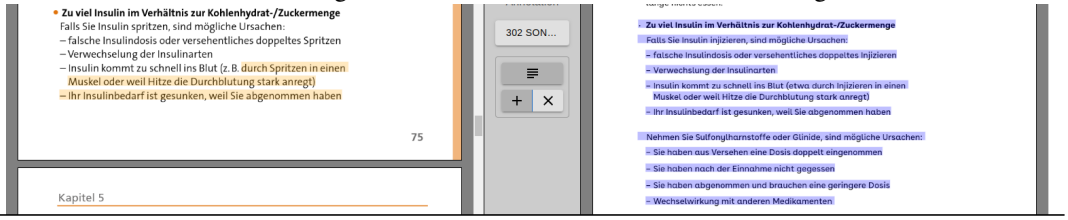
MP8 One entire paragraph describing a drug within the source document was located in the same chapter with other similar paragraphs while in the target chapter this paragraph was located in an isolated position outside of the chapter.

- Such cases unfortunately have to be solved by removing the chapter of the target document with its parent or sibling in the target chapter-inference view and restarting the migration based on those changes.

MP9 Content being entirely gone in the target chapter or additional content added in between.

Also, it was noted that the correctness assumption that each sentence is only annotated once was violated several times. There was one instance were a single sentence in the source document was annotated with six different tags.

Table 6.12: Examples for unmatched content

	Example
MP1	all annotation content differs too much from the original 
MP2	sentence in between other content does not meet threshold 
MP3	rule violation of only full sentences being tagged 
MP4	semantically similar content but BERT embedding similarity not sufficient 
MP5	abbreviations in almost identical text 
MP6	source content split into more or less sentence units 
MP7	source sentence is subset of larger text structure considered a sentence in target 

6.2 Limitations & Future Work

While we attempted to keep the implementation of A^+CHIS^M as generalizable as possible, it still contains some tweaks that optimize it for the tested AOK brochure. It remains yet to be evaluated how A^+CHIS^M performs for other general purpose documents. Especially the Chapter Inference Step implementation has some optimizations for the KE-Chapters. In-between chapters like these will not be attached correctly to the chapter hierarchy by default for other kinds of documents as the application depends on the "Kurz Erklärt" prefix in the chapter title to make the distinction to other content. At the moment all required input and output files of A^+CHIS^M are expected to be used in the context of A^+CHIS. We reason that the implementation of a more generalized tool is possible for arbitrary PDF documents, given a more generalized input format of tagged content. The implementation of a standalone application with the capabilities of the Migration Step for arbitrary PDF documents would be possible if the text locations the text content are provided in an uniform way. Another open question is, how our approach compares to the recently published LLM-based approaches mentioned in Chapter 2. Additionally, linear regression in combination with LLM could be used to determine the ideal threshold parameter values for our employed string comparison techniques for each of the three steps.

7 Conclusion

We describe the implementation of a tool to migrate annotations of unanchored text content within a PDF document to a newer version of that same document. We show that a majority of the required work can be automatized by a combination of algorithmic and machine learning-based approaches. We do this by splitting the task at hand into three distinct sub-tasks, with both an automatic process and an interface where a user may perform manual corrections. We demonstrate the effectiveness of the tool on a use case document with 132 pages (source) and 124 pages (target), respectively which could be migrated within approximately 90 minutes of manual work. The main limitation is that the automatic detection of associated text clusters is not reliable and has to be complemented by manual linking.

Bibliography

[1] *A+chis*, https://apchis.cgv.tugraz.at/, Accessed: March 2024.

[2] F. Moretti, "Conjectures on world literature," *New left review*, vol. 2, no. 1, pp. 54–68, 2000.

[3] S. Lengauer, S. A. Von Götz, M.-T. Hoesch, F. D. Steinwidder, M. Tytarenko, M. Bedek, and T. Schreck, "Guidaeta-a versatile interactions dataset with extensive context information and metadata," *arXiv preprint arXiv:2511.20328*, 2025.

[4] *Aok gesundheitskasse*, https://www.aok.de/, Accessed: December 2025, 2025.

[5] L. Shao et al., "Visual document exploration with adaptive level of detail: Design, implementation and evaluation in the health information domain," in *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2023, Volume 3: IVAPP, Lisbon, Portugal, February 19-21, 2023*, C. Hurter, H. C. Purchase, and K. Bouatouch, Eds., SCITEPRESS, 2023, pp. 133–141. DOI: 10.5220/0011621800003417. [Online]. Available: https://doi.org/10.5220/0011621800003417.

[6] S. Lengauer, M. A. Bedek, C. Kupfer, L. Shao, D. Albert, and T. Schreck, "Recognizing user behavior from interactions for adaptive consumer information systems," in *3rd International Conference on Interactive Media, Smart Systems and Emerging Technologies, IMET 2023, Barcelona, Spain, October 5-6, 2023*, N. Pelechano, F. Liarokapis, A. Asadipour, D. Rohmer, M. Fairén, and J. Moyés, Eds., Eurographics, 2023, pp. 23–26. DOI: 10.2312/IMET.20231251. [Online]. Available: https://doi.org/10.2312/imet.20231251.

[7] *Maxqda*, https://www.maxqda.com/, Accessed: March 2024.

[8] *Qualcoder*, https://qualcoder.wordpress.com/, Accessed: March 2024.

[9] *Pdfbox*, https://pdfbox.apache.org/, Accessed: March 2024.

[10] A. J. B. Brush, D. Bargeron, A. Gupta, and J. J. Cadiz, "Robust annotation positioning in digital documents," in *Proceedings of the CHI 2001 Conference on Human Factors in Computing Systems, Seattle, WA, USA, March 31 - April 5, 2001*, J. A. Jacko and A. Sears, Eds., ACM, 2001, pp. 285–292. DOI: 10.1145/365024.365117. [Online]. Available: https://doi.org/10.1145/365024.365117.

[11] J. J. Cadiz, A. Gupta, and J. Grudin, "Using web annotations for asynchronous collaboration around documents," in *CSCW 2000, Proceeding on the ACM 2000 Conference on Computer Supported Cooperative Work, Philadelphia, PA, USA, December 2-6, 2000*, W. A. Kellogg and S. Whittaker, Eds., ACM, 2000, pp. 309–318. DOI: 10.1145/358916.359002. [Online]. Available: https://doi.org/10.1145/358916.359002.

[12] J. Baumgart, U. Viegener, and C. Pohl, *Den diabetes im griff: Ein handbuch für patientinnen und patienten mit diabetes mellitus typ 2*. Berlin: AOK-Bundesverband, 2021.

[13] *Draftable.com*, https://www.draftable.com/, Accessed: August 2025.

[14] *Diffitdoc.com*, https://diffitdoc.com/, Accessed: August 2025.

[15] *V7labs.com document-comparison-agent 2025*, https://www.v7labs.com/agents/document-comparison-agent, Accessed: August 2025, 2025.

[16] C. Adjetey and K. S. Adu-Manu, "Content-based image retrieval using tesseract ocr engine and levenshtein algorithm," 2021.

[17] L.-C. Chen, H.-T. Weng, M. S. Pardeshi, C.-M. Chen, R.-K. Sheu, and K.-C. Pai, "Evaluation of prompt engineering on the performance of a large language model in document information extraction," *Electronics*, vol. 14, no. 11, p. 2145, 2025.

[18] J. T. Moreira-Filho, D. Ranganath, R. S. Tieghi, R. Patton, V. Sutherland, C. Schmitt, A. A. Rooney, J. Fostel, V. R. Walker, T. Saddler, et al., "Automating data extraction from scientific literature and general pdf files using large language models and knime: An application in toxicology," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 15, no. 5, e70047, 2025.

[19] P. Ingemarsson and P. Daniel, *Pdf parsing, unveiling the most efficient method*, 2024.

[20] T. A. Phelps and R. Wilensky, "Robust intra-document locations," *Comput. Networks*, vol. 33, no. 1-6, pp. 105–118, 2000. DOI: 10.1016/S1389-1286(00)00043-8. [Online]. Available: https://doi.org/10.1016/S1389-1286(00)00043-8.

[21] International Organization for Standardization, *Document management – portable document format*, ISO 32000-1:2008, 2008. [Online]. Available: https://www.iso.org/standard/51502.html.

[22] L. Rosenthol, *Developing with PDF*. O'Reilly Media, Inc., 2013, ISBN: 9781449327910.

[23] V. I. Levenshtein et al., "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, Soviet Union, vol. 10, 1966, pp. 707–710.

[24] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage.," 1990.

[25] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[27] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[29] *Github - easy bert*, https://github.com/robrua/easy-bert, Accessed: June 2025.

[30] *Tesseract*, https://github.com/tesseract-ocr/tesseract, Accessed: October 2025.

[31] *Github - document layout analysis*, https://github.com/huridocs/pdf-document-layout-analysis, Accessed: October 2025.

[32] C. Da, C. Luo, Q. Zheng, and C. Yao, "Vision grid transformer for document layout analysis," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 19 462–19 472.

[33] *Pdf.js*, https://mozilla.github.io/pdf.js/, Accessed: August 2025.

[34] *Spring.io*, https://spring.io/projects/spring-framework, Accessed: November 2025.

[35] *React.dev*, https://react.dev/, Accessed: November 2025.

[36] *Electronjs.org*, https://www.electronjs.org/, Accessed: November 2025.

[37] *Gradle.org*, https://gradle.org/, Accessed: November 2025.

[38] *Apache commons text*, https://commons.apache.org/proper/commons-text/, Accessed: June 2024.

[39] *Opennlp model opennlp-de-ud-gsd-sentence-1.0-1.9.3*, https://opennlp.apache.org/models.html#maven_artifacts, Accessed: June 2024.

[40] *Opennlp.apache.org*, https://opennlp.apache.org/, Accessed: June 2024.

[41] *Bert model multi-cased-l-12-h-768-a-12*, https://www.kaggle.com/models/google/bert/tensorFlow1/multi-cased-l-12-h-768-a-12/1?tfhub-redirect=true, Accessed: June 2025.

[42] M. Neumann, Z. Shen, and S. Skjonsberg, "PAWLS: PDF annotation with labels and structure," *CoRR*, vol. abs/2101.10281, 2021. arXiv: 2101.10281. [Online]. Available: https://arxiv.org/abs/2101.10281.

[43] *React-pdf*, https://react-pdf.org/, Accessed: November 2025.

List of Figures

List of Tables

Acronyms

A$^+$CHIS Advanced interactive, Adaptive, personalized and visual CHIS. 4–8, 11, 16, 26, 35, III

A$^+$CHIS^M A$^+$CHIS Migrator. 6, 8, 9, 11, 12, 22–27, 30, 32, 35, XL

CHIS Consumer Health Information System. 4, 5, 26

CIS Consumer Information System. 4, 5

DSL Domain-specific language. 11

FWF Austrian Science Fund. 4

KE-Chapter "Kurz-Erklärt" - Chapter. 26, 30, 35

LLM Large language model. 7, 8, 35

OCR Optical character recognition. 7, 8

T2DM Type 2 diabetes. 5, 26