Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master’s Research Project

DocProcAI: Leveraging AI for Seamless Content Navigation in MEITREX

Lukas Trautwein, Rick Zolnierek, Valentin Morlock

Course of Study:	Software Engineering
Examiner:	Prof. Dr.-Ing. Steffen Becker
Supervisor:	Niklas Meißner M.Sc.
Commenced:	May 4, 2024
Completed:	November 4, 2024

Abstract

Context. As the world is transitioning to digital and hybrid approaches on teaching and learning, growing amounts of teaching materials are accumulating in learning management systems, giving students unprecedented access to a corpus of documents all at once.
Problem. However, this huge amount of resources also poses significant challenges for learners, as with an increasing number of artifacts such as documents, presentation slides, videos and audio snippets, content discovery and navigation between these artifacts becomes increasingly unwieldy and overwhelming for students, making it necessary for platform providers to offer new ways to inspect, outline, and link materials to enable a more fluent navigation through topically related content.
Objective. In this work we propose and implement a service with which fine-grained semantic information can be extracted from teaching artifacts (lecture slides, documents, videos) and which leverages this information to provide meaningful improvements to the everyday usability for students to allow them to more efficiently navigate, discover, and work with content within the MEITREX intelligent tutoring system.
Method. We explore different techniques from the fields of large language models, digital image processing and natural language processing for the task of extracting a variety of semantic content from teaching artifacts. We explore how this collected data may be used to enhance the user experience of students of digital learning platforms and implement a fully-functioning system based on our concept.
Result. We created a working implementation of the proposed system as a component of the MEITREX learning platform, comprised of a backend service implementing the machine learning pipeline and matching frontend features exposing the new capabilities to the user.
Conclusion. The proposed system successfully enhances content discoverability and usability within digital learning environments, addressing the challenge of navigating large volumes of teaching materials. This work demonstrates the potential of automatic semantic information extraction to enrich the user interface of educational platforms.

Contents

List of Figures

1 Introduction & Motivation

As digital learning is becoming increasingly important, there is an increased focus on creating learning management systems which are more usable and help students learn more easily. However, at the same time, with the increasing abundance of teaching material, navigating through a large corpus of teaching materials is becoming increasingly unwieldy for learners. The large number of documents available can get overwhelming, which significantly reduces usability as it makes it hard for the users to find relevant documents when working on a specific topic, especially when working with multiple types of media at the same time, e.g. lecture notes, video recordings, or exercise sheets.

In contemporary learning management systems – and more broadly in document management systems – this problem is solved by providing a functionality within the system which allows the linking and/or tagging of artifacts, which then allows related artifacts to be linked at appropriate places within the students' user interface. However, in contemporary systems, this tagging and linking process has to be performed mostly by hand, which is quite labor intensive. The linking and tagging usually is also not very fine-grained and can only be performed on a per-document basis.

The MEITREX platform is an e-learning system tailored towards Computer Science Students, created at the Software Quality & Architecture (SQA) group at the Institute of Software Engineering at the University of Stuttgart.

We propose a system within the MEITREX platform which allows for semantic search of video and textual documents, content tagging, linking of relvant sections of documents, summarization of documents, and "chapterization" of videos. Furthermore, the system should perform these tasks automatically with minimal user intervention by using artificial intelligence technologies.

2 Foundations and Related Work

This chapter focuses on foundational knowledge and concepts necessary to understand the project work conducted and explores related work relevant to the project.

2.1 Foundations

This section covers foundational strategies with which text, and in a broader sense documents and other inputs from which text can be extracted, may be processed to gain semantic insights. Additionally, this section provides a brief overview of the specific tools and frameworks which are used or built upon in this work and which are thus necessary to be known to the reader.

Text Embedding Models

Text embedding models allow for the conversion of pieces of text into vectors which represent the semantic meaning of the text piece. The representation of semantic meaning of content as vectors is used as a basis for many natural language processing methods such as semantic search and large language models (LLMs), and conversion of the raw strings to a vector representation is the first step in processing texts in such pipelines.

One way to create a word embedding model is to use neural networks trained using a large dataset of pieces of text[CM21]. A word embedding model trained for generic use can also, in further steps, be fine-tuned to yield more accurate results for a specific topic.

Semantic Similarity of Texts

Semantic similarity describes a measure that indicates how similar two pieces of text are, regardless of how these texts are phrased. Different methods have been proposed to measure the semantic similarity between texts. Knowledge based methods like the one proposed by Corley and Mihalcea [CM05] use manually curated sources like ontologies or dictionaries to extract the meaning of a text. In contrast to that, corpus based methods like [MCS+06] exploit the principle that similar words often occur together in a text, obtaining similarities from large existing sets of text without taking the actual meaning of a text into consideration. Most recently a third category consisting of approaches based on deep neural networks has emerged [CM21].

Semantic Search

Semantic Search is a concept proposed by Guha et al. [GMM03] as an extension of classic search methods which are based on the occurrence of a certain term. It aims to improve the relevance and accuracy of search results by understanding the contextual meaning and relationships of the search terms, rather than relying solely on matching keywords. Semantic search obtains its knowledge from the Semantic Web, an extension of the current World Wide Web, in which information is structured in a way that allows both humans and machines to understand its meaning. While Semantic Search is rooted in the idea of working with human created information, its methods can also be applied to information that's automatically extracted from documents.

As creating semantic connections between content by hand is not practical for the large amounts of data in today's web, nowadays semantic search engines use techniques to automatically derive the semantic meaning of a text and for a given search query then determine the semantic similarity of the text to the search query.

Large Language Models

Large language models (LLMs) are a particular type of machine learning technique which has become popular in recent years. Most state-of-the-art LLMs are based on the transformer architecture. They yield good results for various tasks dealing with creating and processing text and can be used flexibly with free-form text.

Semantic Tagging

Once a document has been transformed into a structured format, semantic tagging can take place. Semantic tagging is the process of creating tags for documents to connect them based on their ontology. More specifically semantic tagging is the extraction and disambiguation of entities and topics in a given text [JBC+14]. Those extracted tags can then be used to compare and link documents with related topics.

Semantic tagging can be performed manually, semi-automatically or automatically, depending on how involved human taggers are. Since the volume of documents and the involved effort has massively increased, automated tagging is a necessity these days.

Topic Modeling

Topic modeling is an unsupervised machine learning approach to find common themes, called topics in large document collections. This is done by finding keywords or phrases that commonly occur together. By automatically creating topics from words and assigning them to documents, they provide a summary overview of a document collection and can assign labels to provide structure to the corpus[LSS+17]. More recently LLM based approaches have also been researched with one of them being BERTopic[Gro22], which converts the documents into word embedding using an LLM and then by clustering them and weighting the clusters creates a topic model. An added benefit to this approach is that it considers semantic information, which traditional topic modeling does not.

Modular Embedded Intelligent Tutoring and Remote Education eXperience (*MEITREX*)

MEITREX [Mei24] is an e-learning platform that integrates various aspects of learning and gamification, designed to be a motivational component within remote or hybrid learning setups. It includes a backend to host all necessary data and a web frontend to present this information to users. The platform offers standard e-learning features such as course pages, course chapters, documents and videos. Additionally, MEITREX incorporates features like quizzes and digital flashcards.

All learning materials can be annotated with various scores which enables students to track their learning progress in detail and to give the system the ability to provide the learner personalized feedback. In addition to that, the scores are used to display leaderboards which are intended to spark competition between learners.

2.2 Related work

Chow et al. [CFCW09] developed a tool to tag the content of lecture slides, ensuring that there was a appropriate number of tags per slide to create groups of and relations between tags to make searching for related content easier. However, they only used the textual content of the slides, compared to our approach which seeks to also include OCR capability for documents and voice recognition for video recordings.

Salimen et al. [SYC+19] performed an analysis of different machine learning methods to automatically tag online news articles for marketing purposes. Similar to our research they used a variety of media types, namely articles and related videos. However, they had a huge number of datasets which were labeled that they could use to train their models.

Baidya and Goel [BG14] proposed a method to create lecture outlines from videos. They segmented videos into topics and added tags to them. Afterwards a comment under the video was created with the timestamps for each topic. However, their method focuses more on segmentation of lecture videos and their suggested keywords were mostly the topics.

3 *DocProcAI* - A Service for Automatic Lecture Material Processing & Enrichment

We propose *DocProcAI*, a service integrated into the *MEITREX* intelligent tutoring system which, upon upload of lecture material, automatically processes it to extract and store relevant semantic information about it.

The processed information is then made available by the service through the unified backend API of the *MEITREX* system for use in the web user interface.

In the *MEITREX* ecosystem, a `Content` refers to an item created in a course by a lecturer.

A `Content` can either be a `Media Content`, which itself consists of one or more documents, videos, etc. – referred to as `Media Records` – which the user has uploaded to their account.

Another type of `Content` is the `Assessment`, of which multiple types exist, currently quizzes and flashcard sets are implemented in the system. An `Assessment` can consist of multiple tasks.

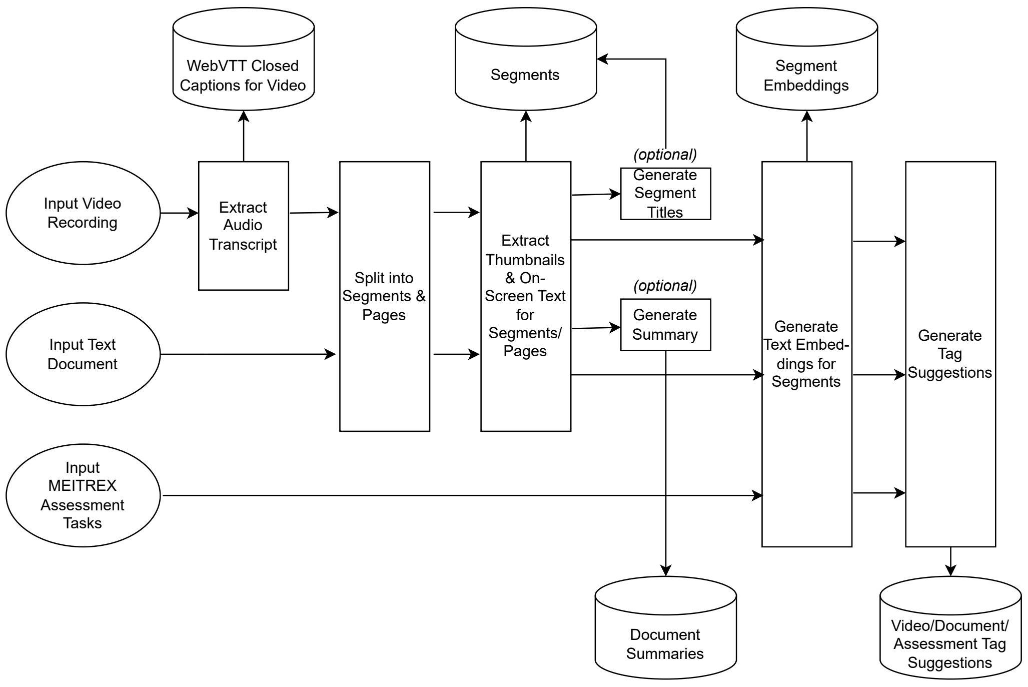

WebVTT Closed Captions for Video
Segments
Segment Embeddings
Input Video Recording
Extract Audio Transcript
Split into Segments & Pages
Extract Thumbnails & On-Screen Text for Segments/ Pages
(optional)
Generate Segment Titles
(optional)
Generate Summary
Generate Text Embed-dings for Segments
Generate Tag Suggestions
Input Text Document
Input MEITREX Assessment Tasks
Document Summaries
Video/Document/ Assessment Tag Suggestions

Figure 3.1: Processing pipeline for the different types of content supported in the *MEITREX* system.

3.1 Document, Video & Assessment Processing Pipeline

The processing of lecture material consists of of a multi-stage pipeline which can be seen in Figure 3.1.

The pipeline can be viewed as three separate sub-pipelines, one for each type of input material. This is necessary as the whole processing sequence varies greatly between the different types as each medium has different properties relevant for its digestion, even though some processing steps are identical for the different types.

3.2 Media Content Item Linking Pipeline

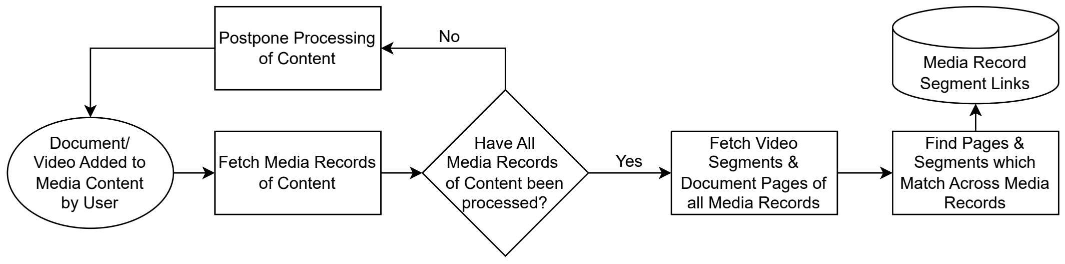

Postpone Processing of Content
No
Media Record Segment Links
Document/ Video Added to Media Content by User
Fetch Media Records of Content
Have All Media Records of Content been processed?
Yes
Fetch Video Segments & Document Pages of all Media Records
Find Pages & Segments which Match Across Media Records

Figure 3.2: Planned media content segment linking pipeline for media content items created for a course in the *MEITREX* system.

Another pipeline is the media content item linking pipeline, which is illustrated in Figure 3.2. It is comparatively simple and its only task is to check the different media records contained in a media content for segments & pages which match – i.e. find the page in a PDF document which is shown in the video at a specific timestamp or vice-versa. It can only be run after the media records contained in the media content have all been processed by their respective pipeline.

For the linking of video segments to document pages, multiple approaches with different advantages and disadvantages were considered.

	Image Template Matching	OCR + Text Semantic Similarity
Pros	• Works for pages only containing images or figures • Threshold can easily be set to produce few false positives	• Works if slide design changed • Works if text changed but semantics stay the same
Cons	• Does not work if slideset design changed between video and document • Does not work if slide text was modified in any way (even just position) • Issues if video heavily edited (webcam, transitions, etc.)	• Does not work for slides containing little text or only figures

In the end, it was decided to use the image template matching approach mainly for two reasons.
Firstly, as for some lectures slides which contain only images or figures with little to no text can be very common, it seemed important to support linking for such slides as well.
Secondly, finding a matching slide via a semantic similarity comparison is conceptually extremely similar to the semantic search function which we already support not only for finding similar documents within the same content item, but course-wide and globally within the user's accessible documents. Thus, it did not seem worthwhile to implement a feature which works on an almost identical basis as an already existing feature and which suffers from the same drawbacks.

4 Implementation

This section describes the implementation of the *DocProcAI Service* and its processing pipeline in detail. The entire processing is contained within the newly created *DocProcAi Service*, which has been integrated into the microservice architecture of the *MEITREX* platform. The service automatically processes any PDF, PPTX or video uploaded to *MEITREX*. Additionally, it processes assessments (quizzes, flashcard sets) created in the web interface of the *MEITREX* platform.

The *DocProcAI Service* is notified of new entities to be processed via PubSub interfaces provided by many other services of the *MEITREX* system. Since the processing tasks can take several minutes to complete, the processing is done in a separate task in the background. For this the documents are added to a processing queue. Ingestion of the media is done in multiple steps, which are described in more detail in the following sections. In case of a system crash, the queue state is kept as a backup in the database of the service, which means that the processing queue can be reconstructed upon service restart and processing can continue.

The processed/generated data of the pipeline is stored within the *DocProcAI Service*'s database, together with the ID of the *MEITREX* media record/content ID of the source document. Processed data can then be accessed through the unified schema provided by *MEITREX*'s graphql gateway by means of schema stitching to inject *DocProcAI Service* data into GraphQL objects returned by the *Media Service*. This allows for a separation of concerns and for the microservices to operate independently of each other.

4.1 Content Extraction & Segmentation

The first step of the processing pipeline involves extraction of the contents of uploaded files into a textual representation. This process differs significantly between video and textual document files.

4.1.1 Video Processing

For videos, in a first step a transcript of the video's audio track is generated using OpenAI's *Whisper* speech-to-text model[RKX+22]. *Whisper* is able to generate transcripts with a sub-sentence granularity, including timestamps of when each snippet of the transcript was said in the source audio. The audio track to be fed to *Whisper* is extracted from the video using *FFmpeg*[Ffm], which allows a wide variety of video formats to be supported.

The output of *Whisper* is used for multiple purposes. First, it allows us to create closed captions using the generated transcript. The output of *Whisper* is converted to the WebVTT closed caption format for later use in the web interface's video player.

Secondly, the transcript snippets provide a first segmentation of the long-form video input. However, these snippets usually have a length of only a few seconds, so they are way too fine-grained. Thus, as a second step, transcript snippets need to be combined into longer, cohesive video segments. To facilitate this, analysis of the video's image data using computer vision is employed.

In a first step, a frame is extracted from the video at each timestamp which marks the beginning of a transcript snippet. For the extraction of video frames, *FFmpeg*'s `select` filters are used. A frame is extracted at a specific second using a filter of the form `lt(prev_pts*TB,{start_time_seconds})*gte(pts*TB,{start_time_seconds})`, where `{start_time_seconds}` is replaced with the start time of the segment. It is necessary to check for both the less-than and the greater-than-or-equal condition because otherwise a frame will not be extracted if the video does not contain a frame within the specified second's timestamp. This can be an issue if the video is encoded with an extremely low frame-rate or with a variable frame-rate.

Then, the extracted images of consecutive transcript snippets are compared using image template matching. For this, *OpenCV*'s image template matching implementation is used.
Whenever the image changes significantly, which is defined by a given similarity threshold, a new segment is created. If, on the other hand, consecutive images are similar, the transcript snippets are merged into longer snippets. Additionally, minimum segment length is artificially limited to 15 seconds. If an image change is detected before 15 seconds have passed, it is ignored. This ensures both that segments are not unreasonably short and that there are not too many segments detected in a video. This artificial limit is especially necessary for video recordings in which the lecturer does not record a slide set but instead is actively drawing on their screen or operating other Software as part of their lecture.
For the image comparison, the extracted video frames are cropped to a centered rectangle of size $(\frac{v_w}{2}; \frac{4*v_h}{5})$, where v represents the videos dimensions in width and height. The cropping is performed due to video recordings possibly containing other elements like letter-boxing, a webcam feed, or copyright watermarks. These elements are usually placed along the edges or corners of the recording, which means they can be easily excluded using cropping so as to prevent them from influencing the accuracy of the image template matching process. For the actual relevant contents of the video screen the cropping has little impact, as the important contents of the video are in almost all cases located towards the center of the screen.

After longer, cohesive segments have been created, on-screen text extraction via optical character recognition (OCR) is performed for the last video frame of each segment. For this, the *Apache Tika*[Apa] content detection framework is used. One notable issue with the matching performance of the OCR framework was its lackluster accuracy for colored text, especially if lighter colors are used. To improve OCR accuracy, the contrast of the images was artificially boosted by a factor of 3, which significantly improved accuracy.

4.1.2 Text Document Processing

For text document processing of files that are not provided in PDF format, the first step is to convert the document to a PDF. This step is necessary because some of the tools used for the following processing steps in the pipeline are only compatible with PDF files. The conversion allows the rest of the pipeline to have a single implementation regardless of the input file type being processed. *LibreOffice*'s[Lib] command line utilities are used for the conversion.

Next, the document is separated page-wise into multiple PDF documents for further processing. Additionally, the document pages are rendered to images, which are made available as thumbnails in the frontend and as part of further processing steps in the pipeline.

The separate pages are then sent to *Apache Tika* for content extraction. *Tika* supports both extraction of text embedded in documents as well as OCR. For PDF documents this means we get the best of both worlds: The text *Tika* returns consists of directly extracted embedded text, if it is available in the document, which does not suffer from the inaccuracies of OCR. For documents which are scans, or for images embedded in the documents, *Tika* automatically performs OCR and also includes the resulting text in its response.

4.1.3 Assessment Processing

Assessment processing is comparatively simple, as only text embedding generation needs to be performed. At the time of writing, the *MEITREX* platform currently supports two types of assessments, quizzes and flashcard sets, although our system is designed to be able to handle arbitrary new assessment types without the need of modification.

This is achieved by the assessment processor using a textual representation of the assessment which is provided by the different microservices which implement the assessments' logic. E.g. for quizzes, the *Quiz Service* provides a basic textual representation of the contents of a quiz, which consists of the questions and answers concatenated for each task. The textual representation can vary significantly between the different types of assessments and is implementation-specific, but this does not pose a problem as the textual representation is only meant to provide a basic way to fetch the contents of an assessment without having to regard any assessment-specific structures.

4.2 Extracting Embeddings

At first the embeddings have to be created as all further operations are based on these embeddings. Depending on the file type (PDF, PPTX or Video), the processes differs.

For lecture slides the slide-set gets divided into individual pages, then the text is extracted. Based on the extracted text the embeddings for each page are created. This allows for more granular control over the process and makes it possible to use PDFs of any length. Furthermore a thumbnail image is created for each page of the PDF. Finally the page is saved to the database as document segment, with its text, the media_record_id it belongs to, the thumbnail, page number and its embeddings.

The video embeddings are generated from both the transcript and the on-screen text. Finally the video segments are saved to the database, with their on-screen text, transcript, media_record_id, start time, a thumbnail, title and the embeddings.

For assesments a textual representation gets used. The embeddings are created from this textual representation.

The embeddings themselves are created by using a SentenceTransformer model, by default 'Alibaba-NLP/gte-large-en-v1.5'. The sentence transformer calculates a fixed-size vector representation of the given text. The embeddings are then used for many of the following steps. Calculating and storing of the embeddings only has to be done once per document, which speeds up all processes relying on the embeddings, as the calculation of the embeddings can be very time consuming. Once the embeddings are calculated the pipeline can proceed.

4.3 Automatic Tag Generation

The tags for videos and documents are generated using a topic model. The topics consist of keywords, which are used in *DocProcAi* as automatically suggested tags. For this all the previously created segments are passed to the topic model. The text of documents, transcript of the videos, or textual representation of the assessments as well as the embeddings are then used to create the model.

BERTopic was used to create the topic model as it is based on the usage of an LLM for creation of embeddings, is modular and has a lot of options to fine-tune the model. It creates the topic model in multiple steps: creation of embeddings, dimensionality reduction, clustering, tokenizing, and weighting. This modularity allows us to use our own embeddings to both speed up the process and keep the results more consistent.

While the default *BERTopic* model already provides good results, it was still necessary to make some adjustments to it. For one the default settings are not well suited for small numbers of documents as is the case here.
Stop words like 'a', 'is', 'the', etc. needed to be removed to improve the quality of the results, in the current iteration this is only done for english, so further adjustments are needed in the future.
Furthermore the ngram range was set from 1 to 3, so that tags could contain short phrases like 'object oriented programming' or 'markov chain', which otherwise would not be possible.
Further adjustments include the reduction of frequent words and the weighting of words to work better on smaller datasets.
Finally the representational model had its Maximal Marginal Relevance adjusted to reduce similar words. A diversity value of 0.3 was chosen, which still provides a large number of relevant keywords while reducing the number of similar keywords. Lower values produced to many similar keywords, while higher values produced to many irrelevant keywords.

It is important to provide enough documents, since otherwise the topic model can't be created. To avoid this issue the topic model isn't run unless at least 11 segments exist in the database. This has proven to be the number of segments where the topic model runs properly and also creates decent results. Adding additional documents increases the quality of the topic model, creating more diverse and accurate topics, while also finding more representative keywords.

One major issue is when the provided text is very short, as this makes it harder for the topic model to assign an appropriate topic. This is primarily a concern with the assessments, as they usually only contain a few short sentences.

Another issue can arise if the documents contain images with a lot of text, as these also get processed by the system. While this is good when the image contains relevant text, if the image is an example that is not directly relevant to the slides it can lead to the topic model using irrelevant keywords from that image. This was the case for one lecture about modeling where the network plan of the VVS was used multiple times in the slides, the topic model decided to use some of the station names as keywords. Rerunning the model or adding more documents can alleviate this issue.

Once the topic model has been created, the representative words for the segments are extracted and combined for each media record or assessment to create a list of suggested tags, which are then saved in the database.

4.4 Automatic Video Segment Title Generation

Video segment titles are generated automatically if title generation is enabled in the *DocProcAI Service*'s config file. Since the title generation uses a large language model, this process can, depending on the model size chosen, be quite expensive in terms of required graphics/system memory and generation can be slow. Thus, if this feature is not needed, it can be disabled. In that case simple counting titles (*Chapter 1*, *Chapter 2* etc.) for the segments and an empty summary are generated.

Titles are generated based on the segments created in the previous steps. The segments' data, constisting of segment start time, transcript and on-screen text, are formatted into a JSON array which is stringified and inserted into the prompt template which in turn is fed to the LLM.

~~~
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a helpful assistant.<|eot_id|><|start_header_id|>user<|end_header_id|>

I have an input JSON file I need to process. It contains an array, where each element is a
snippet of a lecture video. Each element contains the keys "start_time", which denotes the
start time of the snippet in seconds after video start, a "transcript" of the spoken text, and
 "screen_text", the text on screen as detected by OCR. The transcript and screen_text might
contain inaccuracies due to the nature of STT and OCR. The video was split into snippets by
detecting when the screen changes by a significant amount. Please create a JSON object which
contains a property for each segment in the input, where the property key is the start_time of
 the segment, and the value is a string containing your suggested title for that segment.
Choose high-quality and concise titles. If you want two back-to-back snippet to be considered
as the same chapter, give them the same title in your JSON array. Remember to answer only with
 a JSON file. This is the input JSON:

```
{json_input}
```<|eot_id|><|start_header_id|>assistant<|end_header_id|>
~~~

Figure 4.1: Chosen prompt template for the title generation task for a *Llama-3*-family model. For other models, other "Special Tokens" may need to be used in the prompt. `{json_input}` is replaced with the stringified JSON array containing segment data.

While the models used for title generation have been fine-tuned as described in Chapter 5 to make their output conform better to the expected JSON format, this alone cannot ensure with 100% certainty that every output the LLM generates is valid JSON with the expected structure. Solutions for this problem exist, however. We used *lm-format-enforcer*[noa], as it natively supports enforcing LLM output to conform to a given JSON schema.
The JSON format settled on can be seen in Figure 5.3. A format consisting of a JSON array of titles

was also tested. However, the disadvantage of such an array-based format is that it is less strict and verbose. The more verbose format consisting of a JSON object with an explicit property with a key based on the segment's `start_time` yields more consistent title generation results by the LLM. This is explainable by the fact that the LLM does not solely base its next output token on the input given, but also on the output it has previously given. Forcing the LLM to repeat the start time of the segment makes the connection between the outputted title and the inputted segment text clearer and thus leads to better results.

The generated section titles are then passed back and saved in the database.

4.5 Automatic Document Summary Generation

Summaries are generated automatically for uploaded documents if the option is enabled in the service's configuration file.

For the summary generation a large language model is used. Which model and optionally LoRA adapter to use can be controlled in the service's configuration file.

As no format is enforced for the output document summary LLM, the prompt given to the model plays an important role for the style of the generated summary.

~~~
<|begin_of_text|><|start_header_id|>system<|
end_header_id|>

You are a helpful assistant.<|eot_id|><|
start_header_id|>user<|end_header_id|>

The input data:
```
{text_input}
```

I have some text extracted from the document
of a lecture. Please create a very compact
overview/table of contents about which topics
 are covered in this lecture. Your response
must be under 800 characters in length.<|
eot_id|><|start_header_id|>assistant<|
end_header_id|>

Table of contents of the covered topics:
~~~

```
1. Introduction to software architecture and
quality
2. Quantitative properties and measurements
3. Performance evaluation (response time,
throughput, utilization)
4. Reliability and costs
5. Quantitative analysis and modeling
6. Model-driven performance engineering
7. Tool-supported approach
8. Annotated design models and
transformations
9. Performance modeling examples (sequence
diagrams, activity diagrams)
10. Operational quantities and laws
11. Transformations and analysis methods (
queueing networks, stochastic petri nets)
12. Automated architecture optimization
13. Palladio component model and DSL
instances
14. Evolutionary optimization and search
space reduction
15. Multi-criteria optimization and Pareto
frontiers
```

Figure 4.2: Chosen prompt template for the summary generation task on the left and resulting output for a *Llama-3*-family model on the right. For other models, other "Special Tokens" may need to be used in the prompt. `{text_input}` is replaced by the concatenated extracted texts of the pages of the document, separated by `\n\n\n--- Page Break ---\n\n\n`.

4.6 Media Content Linking

Media Records of a Media Content get linked pairwise. For that purpose the thumbnails of the segments of the two media records get compared with each other. Similar to the process for segmentation of videos, image template matching is used to find segments in one media record whose thumbnail matches the thumbnail of a segment in another media record. The threshold for the image template matching function can be set in the configuration file.

Again, *OpenCV*'s image template matching is utilized. However, compared to the image template matching used to segment videos, one major difference is the fact that the size of the image can change in this use case. For example, when exporting presentation slides to a document, many people prefer a layout in which multiple slides are placed on a single page. To also handle this case, simple image template matching is not enough, as it requires the two compared images to be of the same size.
To work around this issue and to enable matching between images of differing sizes, we employ multi-scale image template matching, where multiple image template matching passes are performed, each time with the image being matched against being slightly scaled down.

The other segments thumbnails get resized to the size of the template to improve the quality of matching. All the thumbnails get compared and the one with the highest similiarity gets added to a dictionary. The content linking function also has a similiarity threshold, which must be reached for the linking to take place. Finally all the found matches get saved to the database.

One limitation of the specific image template matching algorithm chosen from *OpenCV* is that it can create some false positives for images which are very barren, e.g. title slides which have a solid-color background and very little text. This issue can be somewhat mitigated by tweaking the matching threshold in the configuration file.

5 Language Model Fine-Tuning & Hyperparameter Tuning

5.1 Segment Title Generation

For the segment title generation task in the content processing pipeline, a large language model was fine-tuned to improve output quality.

Due to limited available hardware resources, the models were quantized to 8-bit representations. Additionally, Low-Rank Adaptation (LoRA) fine-tuning was employed. LoRA fine-tuning is a method by which hardware requirements for training, in particular memory usage, can be significantly reduced. In LoRA fine-tuning, the base model weights are frozen and only a comparatively smaller number of parameters is trained during the fine-tuning process[HSW+21].

Fine-tuning was performed on a machine consisting of an AMD Ryzen 9 7900X, 64 GB of DDR5 memory, and an NVIDIA RTX 3090 graphics card with 24 GB of graphics memory.

5.1.1 Training Data Generation

As base for the training data a total of 134 lecture recordings of 5 courses provided by 3 lecturers were used. The videos had a total cumulative playtime of over 65 hours. The videos were processed using a script which leverages the same code used to extract segments in the production content processing pipeline as described in Section 4.1.1 and store each segment as a JSON object in a file. Another script was used to merge the segments of each video into per-video files consisting of a JSON array of each segment. Each segment of each video was then manually given an appropriate title by a human, resulting in the following JSON structure for the raw training data:

```
[
    {
        "start_time": 123, // Time in seconds when this segment starts in the video
        "transcript": "Transcript of the spoken text in this segment",
        "screen_text": "On-screen text during this segment as recognized by OCR",
        "title": "Appropriate title chosen by a human"
    },
    ...
]
```

Lastly, a script was employed in order to convert the raw training data into prompts which could be fed to the model during training. The *generate_title_training_json.py* script generates training prompts from the input data and a prompt template. For the first training run, the JSON input array consisting of segments $s_1, ..., s_n$ would simply be split up into multiple pieces $s_1, ..., s_k; s_{k+1}, ..., s_l$

etc. in case it was too long to be contained in a single prompt. However, this led to an issue where during inference, the model would oftentimes choose the title "Introduction" for the first segment in its input data, even though this segment was not the first segment in the video (as evident by its `start_time` value). It was concluded that this issue arose because for the vast majority of prompts in the training data, "Introduction" was indeed the first segment's title, as only a comparatively small number of prompts were split up as described above due to excessive length.
To solve this issue, and to increase the amount of prompts in the training data where the first segment is not the beginning of the video, a sliding window approach to training prompt generation was employed for a second training run, i.e. for segments $s_1, ..., s_n$ the prompts $s_1, ..., s_k$; $s_{1+w}, ..., s_l$; etc. are generated, where w is the step size. A prompt always contains as many segements as is possible without exeeding the maximum allowed prompt size. It takes a YAML configuration file which allows customization of the input prompt, maximum length

The prompt template used is identical to the one used during inference which can be seen in Figure 4.1, except for the difference that Control Tokens ("Special Tokens") are omitted from the training prompt, as these are model-specific and inserted automatically by *LLaMA-Factory* depending on the model chosen. The `{json_input}` placeholder is replaced by the input JSON the model is supposed to process.

5.1.2 Model Selection

We chose to employ Meta's *LLaMA-3.1 Instruct* model in its *8-Billion Parameter* version as the baseline model for our tests. A parameter size of at most 8B parameters was fixed due to our hardware constraints, as larger models would require too much VRAM. As comparisons, we also trained a *LLaMA-2 7B Chat* and *LLaMA-3.2 3B Instruct* models on the same corpus of training data.

5.1.3 Training

The *LLaMA-Factory* software[ZZZ+24] was employed to perform the fine-tuning of the models. The reasoning behind this choice was that firstly, *LLaMA-Factory* already allows straightforward configuration of training parameters using a web interface, while at the same time allowing configurations to be stored as YAML files for sharing or to perform subsequent training runs via the command line. More importantly, *LLaMA-Factory* implements many optimization techniques for increased training performance and a reduced memory footprint during training, such as *Unsloth*. These optimizations were crucial to be able to perform fine-tuning on the limited available hardware.

The LoRA technique was chosen to perform the fine-tuning, with the following parameters.
LoRA Rank was set to 128 unless otherwise specified. LoRA rank influences the number of trainable parameters, while other parameters are frozen during training. A higher LoRA rank can result in higher-quality training. The rationale behind choosing a rank of 128 was that for our use case, the model does not need to learn complex contexts, the fine-tuning is mostly needed for the model to generate titles in the expected JSON format. Thus, a rank of 128 is sufficient.

5.1.4 Training Results

In a first training run with 15’000 character long training prompts and no sliding window for prompt generation, both the *LLaMA 3.1* and the *LLaMA 2* model were able to achieve very low training loss results of ca. 0.11 and ca. 0.16 respectively. The models were trained on the dataset for 5 epochs.

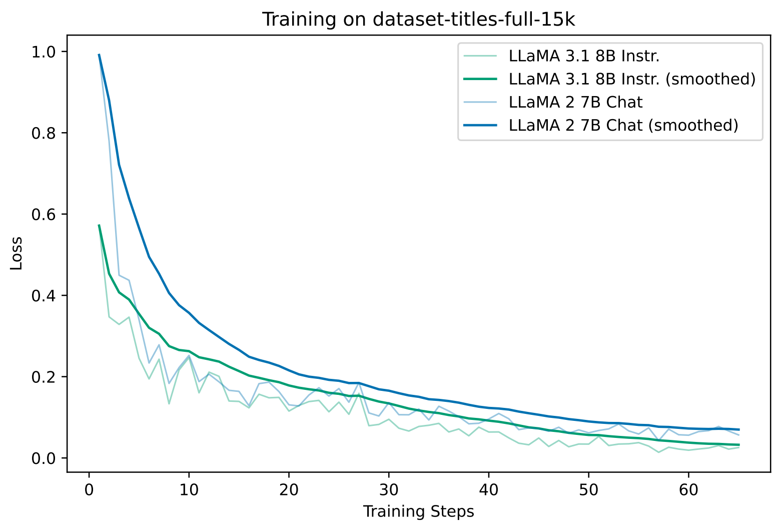


Figure 5.1: Training loss curve for select models on the titles-full-15k dataset.

This is unsurprising, as our corpus of training data is very homogeneous due to the set structure of the JSON output and the predictable nature of title generation.

For a second training run, the prompt length was increased to 20’000 characters, as memory usage during the first training run still had some headroom.

Additionally, the dataset for this and subsequent training runs was created using the sliding window methodology, for the reasons mentioned in Section 5.1.1.
Conversely, the number of epochs the models were trained on the dataset was reduced to 1 to compensate for the increased number of prompts with similar text which were introduced due to the sliding window method of prompt generation. Another model was added for the second training run, namely the smaller *LLaMA 3.2 3B Instruct* model, in order to analyze how smaller, and thus less hardware-resource-intensive, state-of-the-art models can perform in training for this task.

As can be seen in Figure 5.2, results are similar to the first training run, except that training loss has slightly increased across the board. This is to be expected, as through the sliding window prompt generation method, the dataset has become more diverse and in general because we are now training on a larger dataset for 1 epoch instead of a smaller dataset for 5 epochs (although with a very similar number of total training steps in the end).

It can be seen that the *LLaMA 3.2 3B Instruct* model performs only slightly worse than *LLaMA 2 7B Chat*, but with significantly lowed hardware requirements. This encouraged us to explore if the *LLaMA 3.2 3B Instruct* model’s performance could be further improved by increasing the LoRA

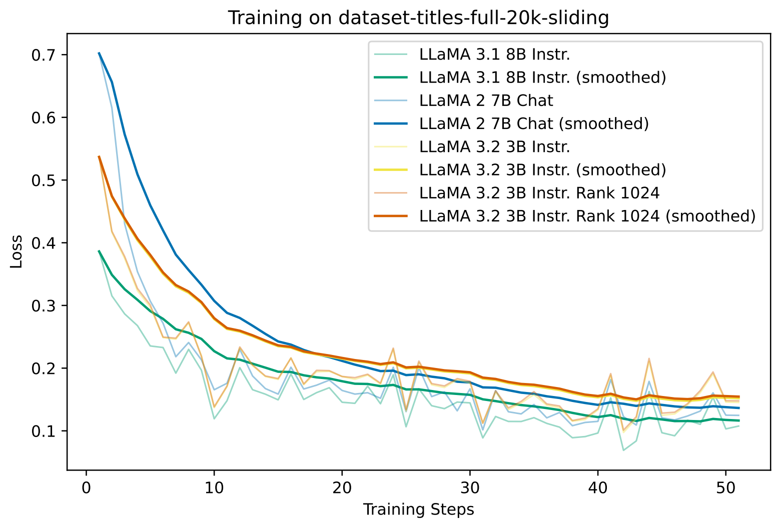


Figure 5.2: Training loss curve for select models on the titles-full-20k-sliding dataset. The yellow line for *LLaMA 3.2 3B Instr.* is hard to see as its curve is almost identical to that of *LLaMA 3.2 3B Instr. Rank 1024*.

training rank in order for it to be on par with the other models. We trained another *LLaMA 3.2 3B Instruct* with a LoRA rank of 1024. However, the increase in LoRA training rank did not have any significant influence on the model's performance, as can be seen in Figure 5.2.

This leads to the conclusion that a LoRA rank of 128 is sufficient in enabling fine-tuning to promote a certain "style" in LLM output – in our case following a JSON schema. This means that large language models can be fine-tuned easily for such tasks even with limited hardware. Such simple fine-tuning tasks seem to not benefit from an increased number of LoRA ranks.

5.1.5 Hyperparameter Tuning

For title generation, the *repetition penalty* parameter plays an important role. As we consider consecutive segments which share the same title to be in the same "chapter" of the video, it is obviously desirable that the LLM repeats the title of the previous segment in that case. For this reason, the repetition penalty must not be set too high. However, at the same time, if the repetition penalty is set too low, the LLM will oftentimes choose the same title for many consecutive video segments, which is undesirable.

Repetition is actually desirable in the use case of segment title generation, as subsequent segments with the same title are considered to be part of the same chapter in the video. Because of this, we set the `repetition_penalty` hyperparameter to 1.0, which equals no repetition penalty.

Sampling was disabled using `do_sample = false`. This is conceptually equivalent to a temperature of 0. Temperature has an influence on the model's output token prediction distribution, where a higher temperature increases the randomness of the model, i.e. the likelihood for the model to pick a token as the next token which it is not the most confident in to be the best fit. A higher temperature

```
{
  "0": "Domain Modeling with UML Class
Diagrams and ER Diagrams",
  "25": "Finding Class Diagrams for the
System",
  "92": "Example Domain: Finding the Class
Diagram Within Given Events",
  "296": "Example Domain: Finding the Class
Diagram Within Given Events",
  "369": "UML Class Diagrams: Reference Card
by Example",
  "543": "UML Class Diagrams: Reference Card
by Example",
  "606": "UML Object Diagrams: Reference Card
 by Example",
  "686": "Example: Domain Modeling (Banking
Domain) - Domain Modeling",
  "785": "Example: Domain Modeling (Banking
Domain) - Domain Modeling",
  "812": "Entity-Relationship-Schema (aka ER
diagram) - Entity-Relationship model"
}
```

```
{
  "0": "Importance of Domain Notion in
Requirements Engineering",
  "25": "Modeling Domains Using Classes and
Relationships",
  "92": "Example Domain: Shopping Shop",
  "296": "Example Domain: Shopping Shop
Continued",
  "369": "Reference Cards for UML Class
Diagrams",
  "543": "UML Class Diagrams: Reference Card
by Example",
  "606": "UML Object Diagrams: Reference Card
 by Example",
  "686": "Example Project: Banking Domain
Model",
  "785": "Example Project: Banking Domain
Model Continued",
  "812": "Entity Relationship Schema (ER
Diagrams)"
}
```

Figure 5.3: Comparison of titles generated with no repetition penalty on the left-hand vs. a higher repetition penalty of 1.15 on the right-hand side.

can lead to more creative outputs, but it can also lead to hallucinations due to the fact that it can make the model pick a choice it is less confident in even though it knows better[HYM+23]. Because of this, we chose a temperature of 0, as hallucinations are highly undesirable in our use case and we do not seek creative output but an accurate recreation of lecture contents for the titles.

5.2 Document Summary Generation

For the summary generation model, no fine-tuning was performed due to the limited hardware resources available.

The rationale behind this decision was that with the available video memory it would not be possible to train the model on a whole document at once. Thus, the document would have to be split up into multiple pieces to allow digestion by the model, which would likely have a negative impact on the quality of the summary.
As the usage of a generic *Llama3.1 8B-Parameter Instruction-Tuned* model already yielded promising results in the summarization task, it was decided that further fine-tuning would be skipped as the creation of the necessary training data would take up a lot of man-hours while at the same time there would be large uncertainties about the quality of the results due to the reasons mentioned above.

5.2.1 Hyperparameter Tuning

As the aim was for the generated summaries to be a very brief overview of the covered topics in the document, thus hyperparameter tuning was employed to reduce the length of the LLM's generated output.
In particular, the *exponential_decay_length_penalty* hyperparameter was used with the values $(300, 2.5)$, where the model is penalized starting at outputs of over 300 tokens in length, with an increasing penalization due to exponential decay of a factor of 2.5.

Additionally, the repetition penalty was set to 1.1 to encourage fewer repetitions in the generated text. A low repetition penality can lead to the model getting stuck in "loops" during inference, where it keeps repeating the same phrase ad infinitum.

As for the hyperparameters of the title generation task, `do_sample = false`, i.e. a temperature of 0, was chosen to disable any sampling. The same reasons as in the title generation task apply for the summarization task.

6 Improving the User Experience Using the Collected Data

This chapter details how the generated data is used in the MEITREX system to enrich the user experience. It describes both the new backend capabilities as well as the new functions and changes to the frontend.

When implementing these features, particular attention was paid to good usability. A common stumbling block in the adoption of smart features is the low-threshold maintenance of the necessary data. Automated processing enables all features to be made available without further intervention by the lecturer.

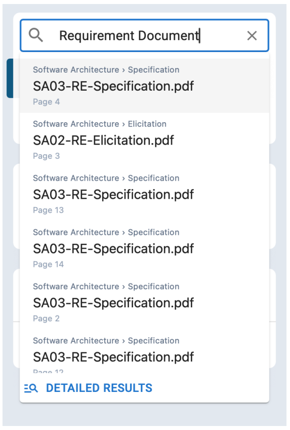
For the end user, i.e. the students, we also placed importance on implementing the new features in a complementary way - this means that students are not forced to adapt to new features, but are provided with a range of options and can progressively integrate those smart functions into their everyday learning life as and when they need them.

6.1 Semantic Search

The semantic search feature allows users to find the segments that are semantically most similar to a query text they input. The number of segments to be returned can be specified by the user. The user can either use the quick search in the sidebar (Figure 6.1) or jump to the detailed search results page (Figure 6.2). The semantic search uses a black- and whitelist, as contents might not be available to students. The query text gets turned into an embedding and then the embedding of the query gets compared to the embeddings of the saved segments. The segments with the highest semantic similarity, measured by lowest distance between embeddings, get returned. How many segments should be returned can be specified by the user. Alternatively a segment can be used as the input. In this case other segments in the same document get ignored and only segments from other documents are considered.

This approach has certain advantages over classic text matching search engine, e.g. the semantic search being able to return documents which match the meaning of the entered search query rather than just returning matches which contain similar text snippets.

The returned segments are then deep linked to the according page of the Meitrex system, e.g. the matching page is directly opened or the playback position of the video is set to the segment the user selected.



Requirement Document

Software Architecture › Specification
SA03-RE-Specification.pdf
Page 4

Software Architecture › Elicitation
SA02-RE-Elicitation.pdf
Page 3

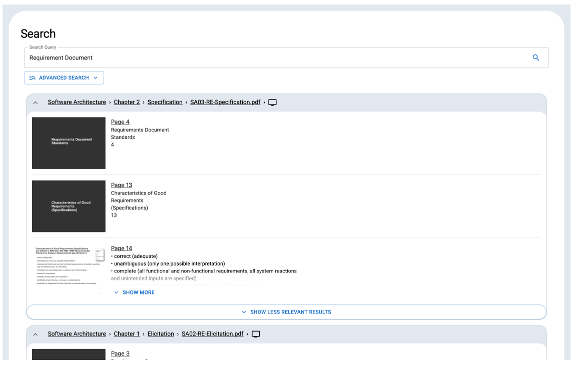
Software Architecture › Specification
SA03-RE-Specification.pdf
Page 13

Software Architecture › Specification
SA03-RE-Specification.pdf
Page 14

Software Architecture › Specification
SA03-RE-Specification.pdf
Page 2

Software Architecture › Specification
SA03-RE-Specification.pdf

Figure 6.1: Semantic search preview within the sidebar, showing entries from the lecture Software Architecture [Stu]



Search

Search Query
Requirement Document

Software Architecture › Chapter 2 › Specification › SA03-RE-Specification.pdf ›

Page 4
Requirements Document
Standards
4

Page 13
Characteristics of Good
Requirements
(Specifications)
13

Page 14
• correct (adequate)
• unambiguous (only one possible interpretation)
• complete (all functional and non-functional requirements, all system reactions
and unintended inputs are specified)

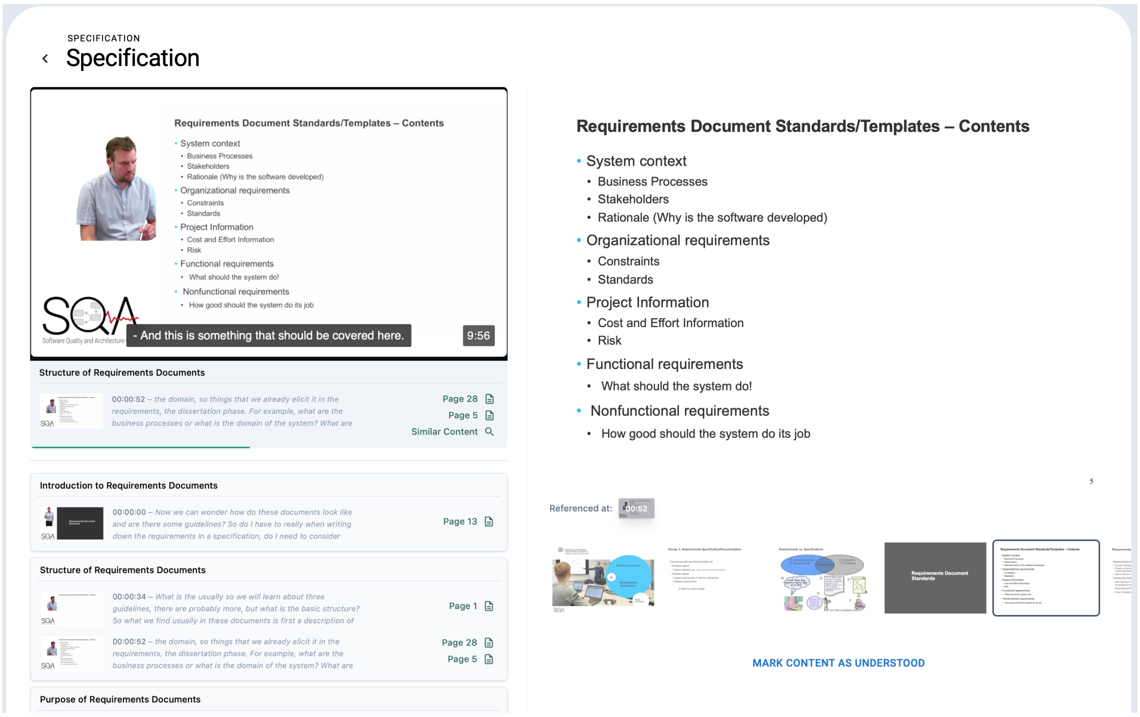
Software Architecture › Chapter 1 › Elicitation › SA02-RE-Elicitation.pdf ›

Page 3

Figure 6.2: Detailed search results, showing entries from the lecture Software Architecture [Stu]

6.2 Split view to view media records side by side

Initially, the content view in MEITREX was limited to displaying a single media file at a time. However, in educational contexts, a common and effective approach to consuming learning materials is to watch lecture videos while simultaneously referencing the corresponding slides, notes, or other documents. This allows users to better contextualize and retain information by linking spoken content directly with visual references. Given this need, we extended the MEITREX system with a split view, as displayed in figure 6.3 that enables videos and documents to be displayed side-by-side. The ratio of the split view is adjustable and we also extended the PDF document viewer by a thumbnail view to be able to navigate between slides more easily.


SPECIFICATION

‹ Specification

- And this is something that should be covered here.

9:56

Structure of Requirements Documents

00:00:52 – *the domain, so things that we already elicit it in the requirements, the dissertation phase. For example, what are the business processes or what is the domain of the system? What are* — Page 28, Page 5, Similar Content

Introduction to Requirements Documents

00:00:00 – *Now we can wonder how do these documents look like and are there some guidelines? So do I have to really when writing down the requirements in a specification, do I need to consider* — Page 13

Structure of Requirements Documents

00:00:34 – *What is the usually so we will learn about three guidelines, there are probably more, but what is the basic structure? So what we find usually in these documents is first a description of* — Page 1

00:00:52 – *the domain, so things that we already elicit it in the requirements, the dissertation phase. For example, what are the business processes or what is the domain of the system? What are* — Page 28, Page 5

Purpose of Requirements Documents

Requirements Document Standards/Templates – Contents

- System context
  - Business Processes
  - Stakeholders
  - Rationale (Why is the software developed)
- Organizational requirements
  - Constraints
  - Standards
- Project Information
  - Cost and Effort Information
  - Risk
- Functional requirements
  - What should the system do!
- Nonfunctional requirements
  - How good should the system do its job

5

Referenced at: 00:52

MARK CONTENT AS UNDERSTOOD

Figure 6.3: Split view with chapter list, Subtitles and PDF-Document on the side, displaying a document and matching video video from the lecture Software Architecture [Stu]

6.3 Automatic subtitles and chapter markers in videos, chapter view

The video player that's part of the aforementioned split view has been extended to display auto generated chapters and subtitles. When searching through the video, the current chapter is displayed in the video progress bar, and the progress bar is annotated by small markers that indicate the beginning of a new chapter. While searching, the extracted thumbnails are displayed as part of the video player's user interface.

The user has the ability to enable subtitles which are generated from the AI transcript and displayed within the video player.

Additionally, a more extensive chapter view is displayed below the video player, displayed in figure 6.4. This view shows a thumbnail and transcript for each section and allows the user to set the video's playback position to the beginning of a certain chapter.

6.4 Automatic linking of videos and slides

Within the split view, a common use-case is to follow the lecturer's video while viewing the corresponding page of the document referenced in the lecture. To make finding the current page easier for the viewer, each video chapter is annotated with links to matching document pages and vice-versa.

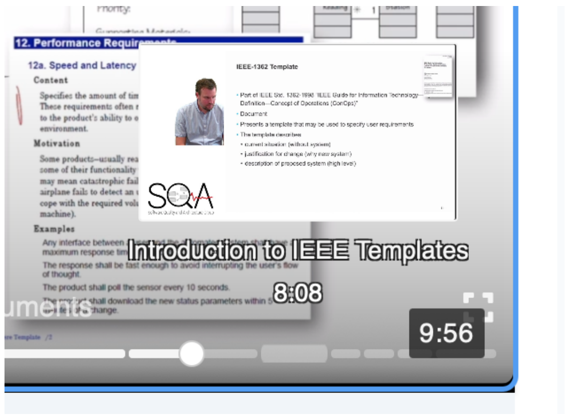


Figure 6.4: Inline chapters displayed by the video player, within a video from the lecture Software Architecture [Stu]

Documents are annotated in the same way, giving the possibility to set the video's playback position to the moment where a certain slide is mentioned.

6.5 Global Search of Similar Content to the Current Video Playback Position

When studying, it can be helpful to be able to find facilitating materials from other documents, either within the same course or in other courses. Using the generated embeddings and semantic search, we've added functionalities to directly search for content that's similar to the currently watched video segment, either within the current course or globally within all courses the user is allowed to view. This "Find similar contents" button opens the same search page which is opened when a user searches for a query and which can be seen in Figure 6.2, but without the user having to manually input a query.

6.6 Global search of similar content for the current quiz question and flashcards

Similar to the global search that has been added to the video player, quiz questions and flashcards are enriched by a link which opens a search results panel that directly displays other content which is related to the currently viewed question or flashcard, without the user having to manually input a search query.

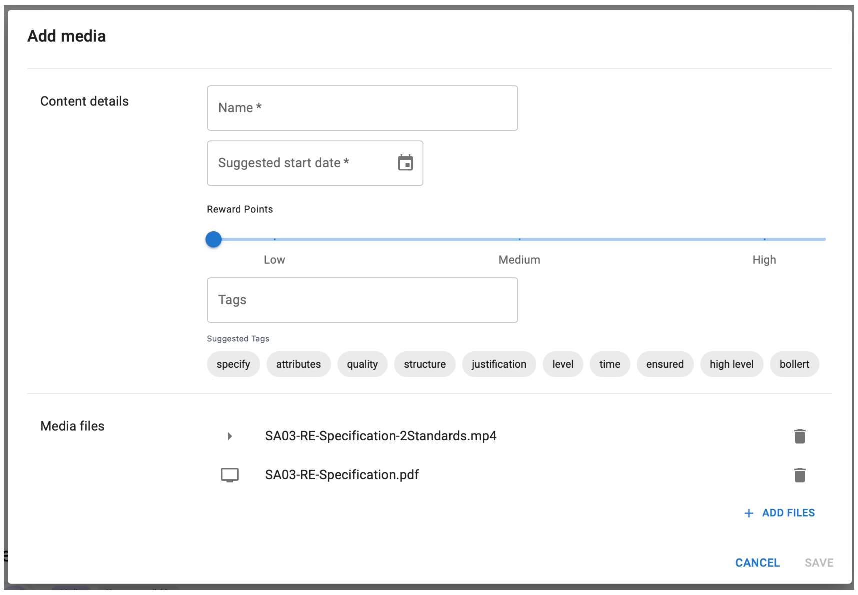


Figure 6.5: Tag suggestions displayed in the content metadata editor

6.7 Suggestion of content tags

After the content has been processed, suggested tags are displayed below the tag input field of the content editor modal (Figure 6.5). In the future, this could be extended by a notification telling the lecturer that new tag suggestions have been found, asking him to re-evaluate the content's tags. Right now, a small badge is shown next to the content link, indicating new tags being available.

6.8 Automatic summaries for uploaded content

The DocProcAI Service automatically summarizes the contents of uploaded documents. The frontend then displays those summaries when the user hovers over a link to a given content on the course page to give the user a quick overview of what's inside the content. This can help the user to navigate the system faster and provides more insights compared to just seeing the title of a document, which in some cases can be uninformative.

7 Evaluation

In this chapter we propose a potential study design for the *DocProcAi* Service. Due to time constraints it was not possible to conduct a user study.

Additionally, as hardware requirements are often an important factor when employing machine learning approaches, and as it was an important goal of this project to develop a system which can run on – for large language models – modest hardware, it is briefly evaluated what minimum hardware is required to deploy the developed *DocProcAI Service* on a machine.

7.1 Potential User Study Design

As the *MEITREX* system was developed to aid computer science students in learning, the study participants would consist of new computer science students.

The study would be separated into multiple parts. Two Interviews or questionnaires, and one practical part.

At first there could be a short questionnaire or interview asking for demographic data, experience in software development, experience with e-learning platforms and expectations regarding e-learning platforms.

The second part would be a practical look at the system. This could consist of tasks that the students would have to solve by using the system. For example students could be asked to find all documents related to a specific document. These tasks would make use of the features added by DocProcAi. How well the tasks were solved could be measured by stopping the time it takes to complete the task and effort to solve the task, which could be measured with a likert scale.

The third part could consist of another interview or questionnaire where students would rate elements of the system on a Likert scale. Criteria here could be: ease of use, logic of generated summaries and titles, how good the links between media files are. Additionally they could be asked qualitative questions about the system, e.g. what additional features would they like, what they liked and disliked.

7.2 Hardware Requirements & Performance

Hardware requirements of the *DocProcAI Service* depend on which features have been enabled in the configuration. Performance was evaluated on a machine consisting of an AMD Ryzen 9 7900X with 64GB of system memory and an NVIDIA GeForce RTX 3090 graphics card with 24GB of video memory.

7.2.1 Document Page Segmentation & Text-Extraction

On the tested machine document segmentation and text extraction took 20 seconds for a document consisting of 80 pages.

7.2.2 Video Segmentation, Audio Transcript Extraction, Text Embedding

Video Segmentation is performed on the CPU, audio transcript extraction and text embedding generation can be run on either the CPU or the GPU if better performance is needed.

Video segment image template matching takes about 25 seconds for a 40 minute video. However, the downloading of the video to a temporary file to process and subsequent extraction of video frames takes an additional 240 seconds.

7.2.3 Large Language Models Tasks

The optional features of title & summary generation use large language models and thus require a GPU to achieve acceptable performance levels at all.

Video segment title generation takes about 65 seconds for a 40 minute video.

Document summary generation takes about 20 seconds for an 80-page document.

Video Memory (VRAM) Requirements

LLMs require, depending on the model size, large amounts of video memory. The model needs to fit into memory to be able to be used at all.

On machines employing *Microsoft Windows*, if video memory is exhausted, up to 50% of system memory (RAM) can also be used as so-called “shared memory”.

The *Linux* kernel does not provide such facilities. This means that the GPU is, in general, limited to using its own video memory.

NVIDIA’s *CUDA* provides its own shared memory implementation which also works on Linux, however this is not supported by PyTorch, which is used by the huggingface *Transformers* library which we use to perform inference. However, attempts have been made by 3rd parties to add support for CUDA Unified Memory to PyTorch[CYK21].

Both video segment title generation and document summary generation require about 14 GB of VRAM when using the *LLaMA 3.1 8B* model.

8 Conclusion

8.1 Summary

In this project machine learning approaches were used to improve the E-learning system MEITREX. More precisely automatic generation of titles and summaries for video segments, automatic generation of suggested tags and linking of related records allow easier exploration of lecture material. This aids students while learning, by showing them relevant related material. The new semantic search feature further helps with finding material that might be related to what the student is currently interested in learning. The new features in the frontend further improve the usability and accessibility of MEITREX, with automatic subtitles and linking of videos and slides. The used machine learning approaches were well suited for the tasks at hand and provided good results.

8.2 Benefits

The main benefit of of our research is to developers of intelligent tutoring systems who want to apply artificial intelligence approaches in their system for summarizing, linking and automated tagging of documents and videos. This paper shows both what is possible and necessary for a solid implementation in this regard and what is required hardware-wise and which limitations need to be accounted for.

8.3 Lessons Learned

The fact that the AI field is moving very fast makes creating an application which does not become outdated fast hard. For example, only days before the project started *LLaMA 3.0* was released. 5 months later *LLaMA 3.2* was already released, which supports vision tasks and could allow significantly improved information extraction from video or document imagery.
It is thus important to create a modular application in which models can easily be swapped out for newer, improved versions. This was achieved by allowing the administrator of the service to easily swap out and configure models using the configuration file. But the example of *LLaMA 3.2* also shows that this is not always feasible: Processing using a vision model would mean a fundamentally different processing pipeline, so a simple swap-out does not work in this case and a significant rework of the application would be necessary in any case.

The quality of the results differs widely based on the model used. It is beneficial to try different models to get a good baseline and the fine tune them further as needed. Furthermore it is a good idea to use different models for different tasks.
When using OCR on images it might be necessary to modify them to achieve better results, e.g.

increasing contrast, cropping, resizing. Most OCR models seem to be optimized for document-format inputs, and less so for presentation slides and similar content.
For LLM based tasks the exact prompt is important, as depending on it the LLM might give undesirable results, e.g. too long responses.

The system works reasonably well on consumer hardware, however a lot of ram and a graphics card with a lot of VRAM are recommended, as the processing time is vastly improved when these are higher. With insufficient resources the PC will also slow down, making it temporarily unusable.

8.4 Future Work

The current system was created to work with english documents and videos. While switching the models to multilingual ones should still work reasonably well, further adjustments might be needed. For example the topic model used to generate the tags only considers english stop words, so it would be necessary to adapt this if other languages are meant to be used.

Expanding the system to support additional filetypes would further improve the usefulness of the system.

Another addition could be the automatic generation of assesments based on the already used machine learning approaches. For example by using LLMs to generate quizzes for the students to solve based on the existing material and quizzes.

With the release of Meta's *LLaMA 3.2-8B Vision* model shortly before the deadline for this project, smaller-scale vision models with lower hardware requirements have hit the mainstream. Our current approach uses text extraction from documents with OCR, which makes semantic extraction from graphical content like charts and diagrams impossible. Usage of a multi-modal LLM may enable the system to extract even more valuable semantic information from the documents it is provided. For this to work however, a rework of the currently implemented pipeline would be necessary.

A more extensive evaluation of the created system via a user study is planned for the near future.

8.5 Source Code Repository & Training Data

The source code of the DocProcAI service is made available at its GitHub repository[Doca] as part of the open-source code of the whole *MEITREX* ecosystem.

Scripts for the generation and processing of the training data as described in Chapter 5 are made available in a separate repository.[Docb]

Raw training data and the trained models are made available upon request.

Bibliography

[Apa] *Tika*. https://tika.apache.org/. Apache Foundation (cit. on p. 12).

[BG14] E. Baidya, S. Goel. “LectureKhoj: Automatic tagging and semantic segmentation of online lecture videos”. In: *2014 Seventh International Conference on Contemporary Computing (IC3)*. IEEE, Aug. 2014. DOI: 10.1109/ic3.2014.6897144 (cit. on p. 5).

[CFCW09] K. O. Chow, K. Y. K. Fan, A. Y. K. Chan, G. T. L. Wong. “Content-Based Tag Generation for the Grouping of Tags”. In: *2009 International Conference on Mobile, Hybrid, and On-line Learning*. IEEE, Feb. 2009. DOI: 10.1109/elml.2009.22 (cit. on p. 5).

[CM05] C. D. Corley, R. Mihalcea. “Measuring the semantic similarity of texts”. In: *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*. 2005, pp. 13–18 (cit. on p. 3).

[CM21] D. Chandrasekaran, V. Mago. “Evolution of semantic similarity—a survey”. In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–37 (cit. on p. 3).

[CYK21] J. Choi, H. Y. Yeom, Y. Kim. “Implementing CUDA Unified Memory in the PyTorch Framework”. In: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. 2021, pp. 20–25. DOI: 10.1109/ACSOS-C52956.2021.00029 (cit. on p. 32).

[Doca] *DocProcAI Service*. https://github.com/MEITREX/docprocai_service. MEITREX (cit. on p. 34).

[Docb] *DocProcAI Training Scripts*. https://github.com/MEITREX/docprocai-training. MEITREX (cit. on p. 34).

[Ffm] *FFmpeg*. https://ffmpeg.org/. FFmpeg Team (cit. on p. 11).

[GMM03] R. Guha, R. McCool, E. Miller. “Semantic search”. In: *Proceedings of the 12th international conference on World Wide Web*. 2003, pp. 700–709 (cit. on p. 4).

[Gro22] M. Grootendorst. “BERTopic: Neural topic modeling with a class-based TF-IDF procedure”. In: (Mar. 2022). DOI: 10.48550/ARXIV.2203.05794. arXiv: 2203.05794 [cs.CL] (cit. on p. 4).

[HSW+21] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685 (cit. on p. 19).

[HYM+23] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, T. Liu. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2023. arXiv: 2311.05232 [cs.CL]. URL: https://arxiv.org/abs/2311.05232 (cit. on p. 23).

[JBC+14] J. Jovanovic, E. Bagheri, J. Cuzzola, D. Gasevic, Z. Jeremic, R. Bashash. "Automated Semantic Tagging of Textual Content". In: *IT Professional* 16.6 (Nov. 2014), pp. 38–46. ISSN: 1941-045X. DOI: 10.1109/mitp.2014.85 (cit. on p. 4).

[LSS+17] T. Y. Lee, A. Smith, K. Seppi, N. Elmqvist, J. Boyd-Graber, L. Findlater. "The human touch: How non-expert users perceive, interpret, and fix topic models". In: *International Journal of Human-Computer Studies* 105 (Sept. 2017), pp. 28–42. ISSN: 1071-5819. DOI: 10.1016/j.ijhcs.2017.03.007 (cit. on p. 4).

[Lib] *LibreOffice*. https://www.libreoffice.org/. The Document Foundation (cit. on p. 12).

[MCS+06] R. Mihalcea, C. Corley, C. Strapparava, et al. "Corpus-based and knowledge-based measures of text semantic similarity". In: *Aaai*. Vol. 6. 2006. 2006, pp. 775–780 (cit. on p. 3).

[Mei24] N. Meißner. "MEITREX-Gamified and Adaptive Intelligent Tutoring in Software Engineering Education". In: *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. 2024, pp. 198–200 (cit. on p. 5).

[RKX+22] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022. arXiv: 2212.04356 [eess.AS]. URL: https://arxiv.org/abs/2212.04356 (cit. on p. 11).

[SYC+19] J. Salminen, V. Yoganathan, J. Corporan, B. J. Jansen, S.-G. Jung. "Machine learning approach to auto-tagging online content for content marketing efficiency: A comparative analysis between methods and content type". In: *Journal of Business Research* 101 (Aug. 2019), pp. 203–217. ISSN: 0148-2963. DOI: 10.1016/j.jbusres.2019.04.018 (cit. on p. 5).

[Stu] U. of Stuttgart. *Software Architecture*. Lecture. Held at the University of Stuttgart (cit. on pp. 26–28).

[ZZZ+24] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, Z. Luo, Z. Feng, Y. Ma. "LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Bangkok, Thailand: Association for Computational Linguistics, 2024. URL: http://arxiv.org/abs/2403.13372 (cit. on p. 20).

[noa] noamgat. *lm-format-enforcer*. https://github.com/noamgat/lm-format-enforcer (cit. on p. 15).

All links were last followed on November 4, 2024.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature