

Abstract

In recent years e-learning platforms and online courses are spreading more and more, also boosted by the recent COVID-19 pandemic. However, all the most popular platforms tend to represent the material in a linear way, without exploiting the relations between the topics explained in the video. The current method of fruition of the educational videos can be re-discussed using knowledge extracted from the video itself in order to facilitate the learning process of a student.

With this project, called EDURELL, we aim to deepen the relations between concepts explained in an educational video in order to reduce the workload of a learner and to offer a different experience for browsing the video.

Therefore, the main field of research of this system is how to extract the prerequisite relations, which are the most important type of relation between concepts in an educational video because they show the learning order of a new topic. The prerequisite relations allow also the creation of a concept map, in which each node is a concept and each link is a relation, that can be used to augment a video in order to improve the user experience in learning new topics.

This system offers an easy to use interface for the manual extraction of prerequisite relations, and, since manual annotation can be time consuming, it offers an automatic method of extraction which can be tested by comparing the results with the manual extraction. In this dissertation, the results of the EDURELL automatic method are compared with three baseline methods of the current state of art, and it shows an improvement in different metrics such as F1-score, Vertex Edge Overlap and Cohen's kappa.

The obtained knowledge graphs are then organized using the Web Annotation ontology, expanded where needed, and stored in an online database, in order to permit the exploitation of the knowledge graph for the video augmentation for learners.

Contents

1	The EDURELL system	1
1.1	Goals and Description of the System	1
1.2	System Architecture	2
1.3	Tools and libraries	4
1.4	Ontology-based Data Model	7
1.5	Database	11
1.5.1	Examples queries and performances	13
1.6	Semi-automatic Video Annotation Tool	15
1.6.1	Goals and description	15
1.6.2	Video Preprocessing	16
1.6.3	Video annotation	22
1.6.4	Analysis	28
1.6.5	Burst Analysis and temporal reasoning	30
1.6.6	Evaluation	37
1.7	Video augmentation for learners	47
1.7.1	Goals and description	47
1.7.2	Video Exploration	48
1.8	Server deployment	52

Chapter 1

The EDURELL system

In this document, we illustrate the EDURELL system and all its features.

The first section contains the goals that we aim to achieve, then the architecture of the system is explained with some diagrams to make the process easier. In the third section we list all the main tools and libraries used for the implementation. In the fourth and fifth sections we describe the database, and in particular, which is the data model that is implemented - with some examples.

Then, we describe both the video annotation and the video augmentation, also showing the interfaces of the system and the results obtained.

Finally, we explain how the system is deployed on the server

1.1 Goals and Description of the System

Concept maps are a way to represent key concepts by following the learning order. In this representation each node is a concept and the links between them are the prerequisite relations.

Therefore, the use of concept maps can indicate to the student the learning path in order to fully apprehend a new concept.

The EDURELL project aims to facilitate the learning process of a student watching an educational video by taking advantage of concept maps and by pointing out where a concept is described in the video.

In order to achieve these purposes the system must first permit the extraction of the knowledge graphs, which it can be done manually by an expert annotator, semi-automatically or automatically by exploiting the Burst analysis that we will explain in section 3.6.5.

The knowledge graphs, that contains the prerequisite relations and the concepts descriptions, tracked in terms of time in the video, are then used to augment the video, and thus, to allow a learner to browse the video in a new scenario, optimizing the learning process in terms of time and user experience.

1.2 System Architecture

The architecture of the system is composed from two tools.

The first tool "*EDURELL Annotation Tool*" is designed for the extraction of knowledge graphs from educational videos.

The videos are taken from Youtube and, in order to extract all the prerequisite relations and all the points where a concept is explained, the video is processed through Python Flask, which it will organise the concept map in a knowledge graph that is stored on MongoDB Atlas in a JSON-LD format. Python Flask allows us also to provide several type of analysis of the extracted graphs.

The second tool "*EDURELL Video Augmentation*" is designed to support learners in the use of educational contents in multimedia format, through the hypervideo functionalities. The aim is to improve the browsing experience, and to receive support during the use of educational videos, by taking advantage of knowledge graphs extracted with the EDURELL Annotation Tool.

Both of the tools uses Python Flask as back end and they share the same MongoDB database, for the front end part however, the first tool is implemented using Javascript and Bootstrap, and the second one using the React technologies.

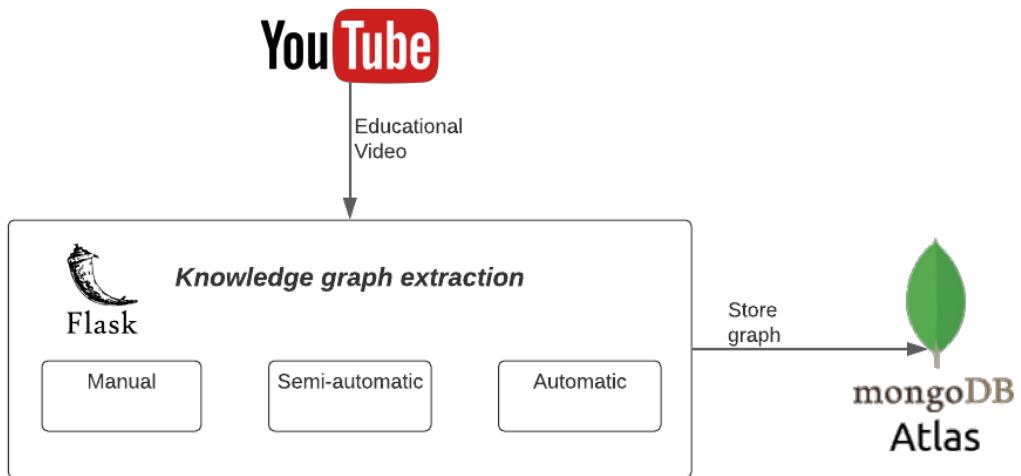


Figure 1.1: Architecture of the annotation tool

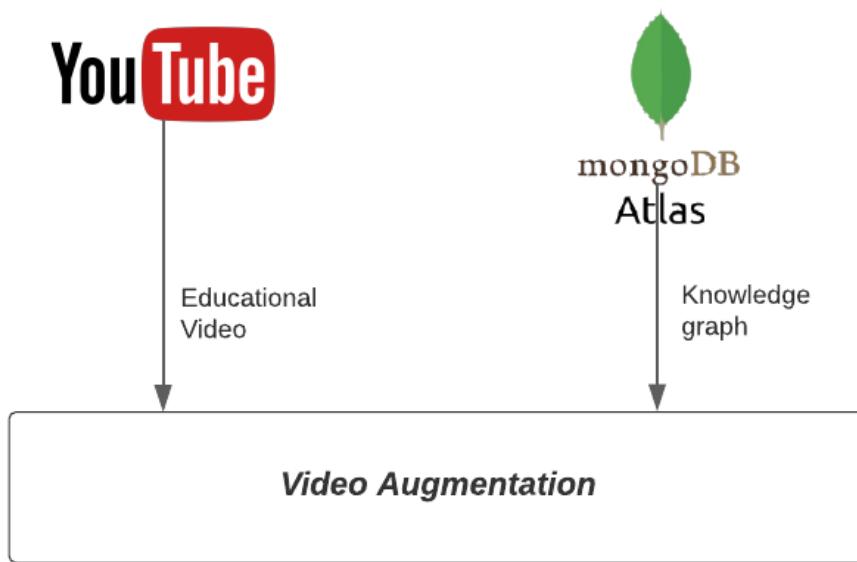


Figure 1.2: Architecture of the video augmentation tool

1.3 Tools and libraries

This section describes the main tools and libraries used for the implementation of the project.

Python Flask

A web application is needed in order to permit an easy access to the system through a browser. To achieve this purpose the project is implemented by using Flask¹, which is a web framework written in python. Moreover, Python will allow us to use all libraries needed.

Youtube-dl and youtube-transcript-api

Youtube-dl² is a Python package that allows to download videos from YouTube.com and other platforms.

Youtube-transcript-api³ is a python API which allows you to get the transcript/subtitles for a given YouTube video. It also works for automatically generated subtitles and it supports translating subtitles.

Punctuator

Punctuator⁴ is a bidirectional recurrent neural network model that allows to add punctuation in a text. For the periods, which are our main goals, the score of the pretrained model are precision 72.3, recall 71.5 and F1-score: 71.9

Sentence-transformers

SentenceTransformers⁵ is a Python framework for state-of-the-art sentence, text and image embeddings.

¹<https://flask.palletsprojects.com/en/2.0.x/>

²<https://github.com/ytdl-org/youtube-dl>

³<https://pypi.org/project/youtube-transcript-api/>

⁴<https://github.com/ottokart/punctuator2>

⁵<https://github.com/UKPLab/sentence-transformers>

This framework can be used to compute text embeddings. These embeddings can then be compared e.g. with cosine-similarity to find sentences with a similar meaning. This can be useful for semantic textual similarity.

The framework is based on PyTorch and Transformers and offers a large collection of pre-trained models tuned for various tasks.

Pandas

Pandas⁶ is a data analysis tool built with Python, it allows a fast data manipulation through the use of the pandas dataframes.

Pytorch

Pytorch⁷ is an open-source Python library used mainly in machine learning and deep learning, it is specialized in tensor computations, automatic differentiation, and GPU acceleration.

OpenCV

OpenCV⁸ (Open Source Computer Vision Library) is an open source computer vision library, it allows a real-time image processing with different implementations of computer vision algorithms. It has different programming interfaces like C, C++, Python and Android.

Rdflib

RDFLib⁹ is a Python package to manage RDF triples. It contains Parsers and Serializers (RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, JSON-LD) and it allows to perform query SPARQL.

⁶<https://pandas.pydata.org/>

⁷<https://pytorch.org/>

⁸<https://pypi.org/project/opencv-python/>

⁹<https://rdflib.readthedocs.io/en/stable/>

PyMongo

PyMongo¹⁰ is a Python distribution containing tools for working with MongoDB, it allows to upload and query the data of the MongoDB Atlas database.

Nltk

Nltk¹¹ (Natural Language Toolkit) is a Python package that provides a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

Conllu

CoNLL is a Tab Separated Value data format, used in Natural Language Process, in which each line is a word and each column is an attribute of the word such as: ID (index in sentence), FORM (word form or punctuation symbol), LEMMA (lemma or stem of word form), UPOS (universal part-of-speech tag), XPOS (Language-specific part-of-speech tag)

Conllu¹² it's a Python library that allows to parse and to manage conll format data.

Bootstrap

Bootstrap¹³ it's a Javascript front-end library that allows a quick and easy design of responsive web apps.

React

React¹⁴ it's a Javascript front-end open source library for building user interfaces created by Facebook.

¹⁰<https://pymongo.readthedocs.io/en/stable/>

¹¹<https://www.nltk.org/>

¹²<https://pypi.org/project/conllu/>

¹³<https://getbootstrap.com/>

¹⁴<https://it.reactjs.org/>

Vis.js

Vis.js¹⁵ is a Javascript visualization library that allows the creation of networks. The library supports also custom styles and allow to interact with the visualized graph. The network visualization works smooth on any modern browser for up to a few thousand nodes and edges and it uses HTML canvas for rendering.

1.4 Ontology-based Data Model

Educational videos represent one of the most popular formats in online education, but they can also suffer of several limitations. They can for instance be lengthy and thus hard to navigate when you need to recall concepts; besides, they generally lack explicit tables of contents or other indexing systems for exploring the video content.

To solve this problem, the first step is to have an effective representation of the content of the video, with a data model which is able to encode both concepts which are explained in it and evolving relations between them, relating concepts with their prerequisite (weak or strong).

For this purpose we used the Web Annotation Ontology, extended where needed, in such way to exploit the semantic web benefits and obtain an explorable knowledge graph, in json-ld format. The Edurell Data Model¹⁶ describes a structured model to annotate video resources with the concepts which are explained and with prerequisite relationships between those concepts.

There are three types of annotation:

- CoNLL Annotation: Each video resource may be associated with at most one ConLL file which is used to describe the words of the video transcription.

As motivation of the annotation is used *edu:linkingConll* which extends the original *oa:linking* motivation, as in figure 1.3.

¹⁵<https://visjs.github.io/vis-network/docs/network/>

¹⁶<https://frcassi.github.io/edurell/>

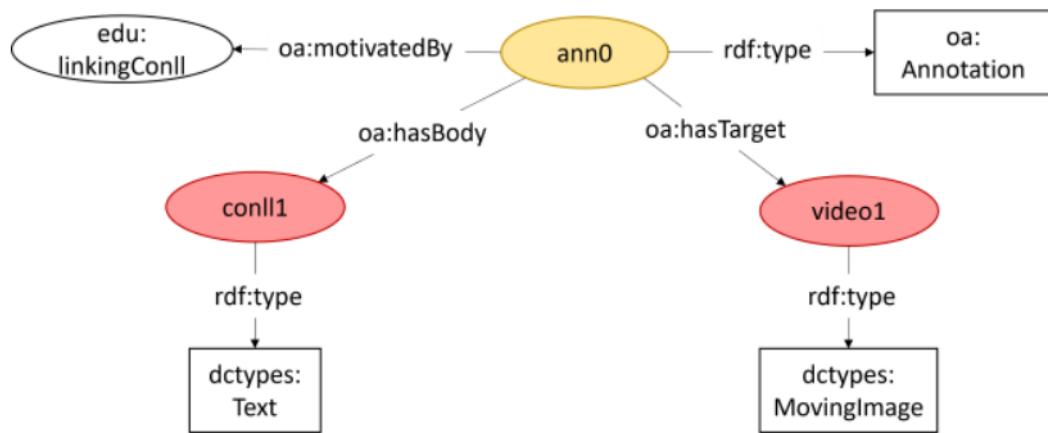


Figure 1.3: CoNLL linking

- Concept description: in this annotation we need to have the start and the end time in which a concept is explained, the corresponding sentences of the CoNLL and if the description is a “definition” or a “in depth” explanation.

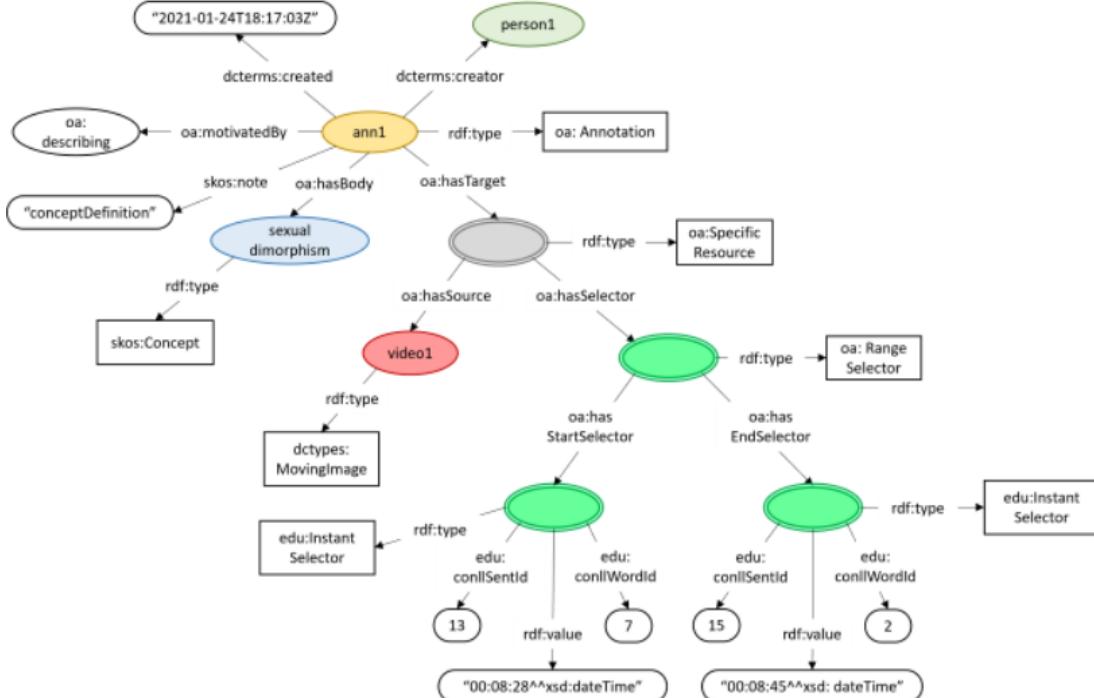


Figure 1.4: Annotation of a description

The body of the annotation is the concept that is being explained, which is of the type *SKOS:Concept*. The target of the annotation is given by an empty node which links the video and the selector to have start and end time of the concept's description. The video is linked by the web annotation ontology property *oa:hasSource*, which explains that the description has as source the video.

In order to have the times, two selectors from oa ontology are used, and in order to have also the link to the CoNLL file, we extended the ontology with *edu:conllSentId* and *edu:conllWordId* that respectively contains the id of the sentence in the conll and the id of the word.

In order to distinguish between definition of a concept (description which contains the main definition of the concept) and in depth description (description of the concept that adds info to the concept but it's not the main definition) the *SKOS:note* property is used.

- Prerequisite relations: this annotation includes the prerequisite and the target of the relation, the weight (strong or weak), the starting time, the portion of the frame that contains the target (if present) and the id of the CoNLL's sentence. An example is given in figure 1.5

In this annotation the body is given by the *SKOS:Concept* which is the prerequisite of the relation. Similarly to the concept's description annotation, the target of the annotation is given by an empty node with source the video, but it has a property *dcterms:subject* with the concept which is the target of the prerequisite relation.

The Web Annotation's selector contains the starting time of the prerequisite relation using *rdf:value*, the ids of the sentence and the word in the CoNLL, and the portion of the video's frame in which there is the target concept, given by the property *edu:hasMediaFrag*.

In order to distinguish between strong relations (prerequisites that are strictly needed for the understanding of the target concept) and

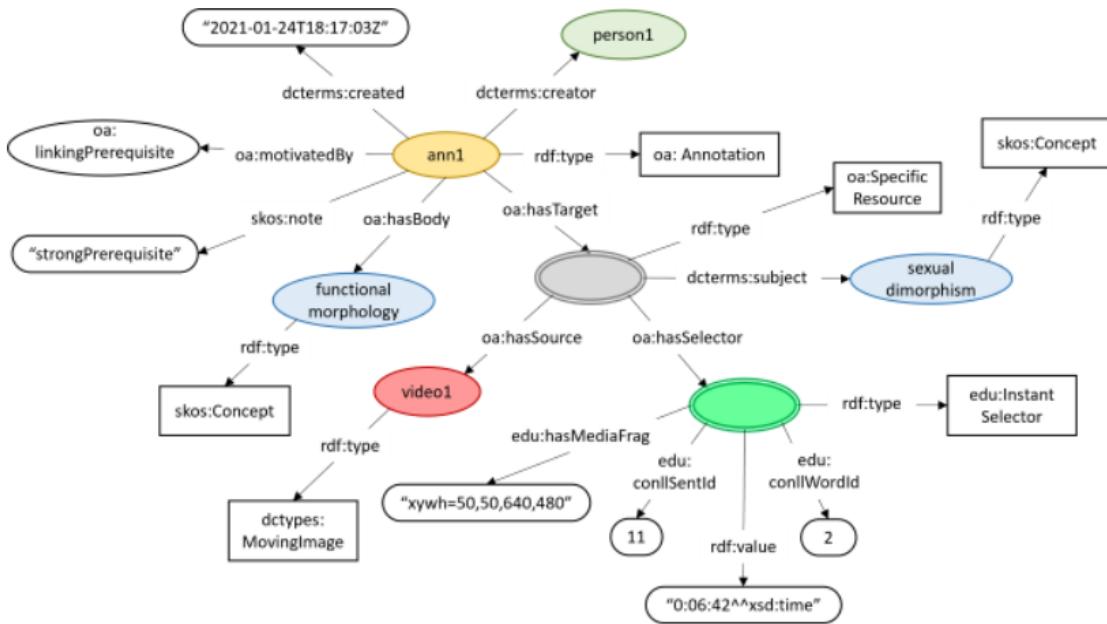


Figure 1.5: Annotation of a prerequisite relation

weak relations (not strictly needed but useful) the *SKOS:note* property is used.

In order to easily distinguish between the different types of annotations the *oa:motivatedBy* property is used. For a concept description the annotation is motivated by *oa:describing* and for a prerequisite relation the annotation is motivated by *edu:linkingPrerequisite*, which extends the original *oa:linking*.

1.5 Database

To store JSON data for each input video, we explored a NoSQL database solution. This technology stores documents directly in JSON format and thus is very suited for our application.

The two main contenders for this technology are Firebase and MongoDB. Firebase is a collection of cloud services offered by Google, while MongoDB is just an open-source technology that you can use, and thus is fairer to compare Firebase to MongoDB Atlas.

Firebase offers two cloud database services and they are both hosted directly by Google. The two databases are Firestore and Realtime Database.

According to Google survey¹⁷ the second option suits our needs better because we want to save data as simple JSON trees. But it has several limitations, for example it's not possible to filter on multiple conditions, and thus we optioned for the MongoDB Atlas database which permits an easy query of the knowledge graphs with high performances.

MongoDB collections

- Videos: this collection stores all the video's data such as the youtube id of the video, the title, creator, and all the data extracted through the video processing that we will explain in section 3.6.2
- Conlls: it contains all the CoNLLs and corresponding ids
- Graphs: annotations extracted, it includes the ids of the annotator and the JSON-LD of the annotation
- Users: the users are common to both the applications, this collection contains name and surname of the user, email, hashed password if the account is validated or not, and the history of the previous videos watched.

¹⁷<https://firebase.google.com/docs/database/rtdb-vs-firebase>

```
_id: ObjectId("60e4352b53d2c706e7d6e1e1")
name: "Luca"
surname: "Mirenda"
email: "mirenda_luca@yahoo.com"
password_hash: "$2b$12$QYBRpVAn3Z31NP5y.ZFZwuM/VJJvX7DkGcGMd70uz
✓ video_history_list: Array
  ✓ 0: Object
    video_url: "https://www.youtube.com/watch?v=sXLhYST00m8"
    video_watchtime: 408
    fragment_clicks: 18
    node_clicks: 13
    transcript_clicks: 1
    searchbar_clicks: 3
    notes: "aa"
    lastChangesDate: 2021-11-10T17:48:02.307+00:00
    > fragments_progress: Array
    > logs: Array
  > 1: Object
  > 2: Object
  > 3: Object
  > 4: Object
```

Figure 1.6: Users collection

For each video seen by the user all the necessary data is stored, such as the watching time of the video in order to permit to the user to restart the video from where he left and the notes that the student took during the lesson.

1.5.1 Examples queries and performances

In order to query the database we tested two approaches.

The first approach is to get the json from the database and to parse it using the rdflib library, then we retrieve the information using a SPARQL query.

```

1 from rdflib import Graph
2 import time
3 import json
4
5 def get_descriptions(json_graph):
6
7     gr = Graph() \
8         .parse(data=json.dumps(json_graph), format='json-ld')
9
10    tic = time.time()
11
12    # Get all concepts explained in the video
13    query1 = """
14        PREFIX oa: <http://www.w3.org/ns/oa#>
15        PREFIX edu: <http://edurell.com/>
16        SELECT ?explained_concept
17        WHERE {
18            ?ann oa:motivatedBy oa:describing .
19            ?ann oa:hasBody ?explained_concept .
20        }"""
21
22    qres = gr.query(query1)
23
24    toc = time.time()
25
26    return qres

```

Code 1.1: SPARQL query

In Code 3.1 there is an example of this approach, in which we identify all the concepts that have been explained in the video. Results on big graphs (100 or more prerequisite relations and concepts definitions) show an average of 0.7 seconds in order to obtain the data.

The second approach consists to get the data directly from MongoDB Atlas, by querying the data using the PyMongo library.

```

1 def get_concept_map(annotator, video_id):
2     collection = db.graphs
3
4     pipeline = [
5         {"$unwind": "$graph.@graph"}, 
6         {"$match": {
7             "video_id": str(video_id),
8             "annotator_id": str(annotator),
9             "graph.@graph.type": "oa:annotation",
10            "graph.@graph.motivation": "edu:linkingPrerequisite",
11        }},
12    ],
13    ,
14
15    {"$project": {
16        "prerequisite": "$graph.@graph.body",
17        "target": "$graph.@graph.target.dcterms:subject.id",
18        "weight": "$graph.@graph.skos:note",
19        "time": "$graph.@graph.target.selector.value",
20        "sent_id": "$graph.@graph.target.selector.edu:conllSentId",
21        "word_id": "$graph.@graph.target.selector.edu:conllWordId",
22        "xywh": "$graph.@graph.target.selector.edu:hasMediaFrag",
23        "creator": "$graph.@graph.dcterms:creator",
24        "_id": 0
25    }
26    },
27    ,
28
29    {"$sort": {"time": 1}}
30]
31
32 aggregation = collection.aggregate(pipeline)
33 concept_map = list(aggregation)
34 return concept_map

```

Code 1.2: PyMongo query

In Code 3.2, the query gets the concept map annotated by a specific

annotator for a video, and thus all the prerequisite relations with all the related data, such as the time for which the relation starts and the relative sentence and word ids of the CoNLL.

Results on big graphs show an average of 0.02 seconds to get the data.

For this reason, the tests show that it is faster to query directly the database through PyMongo instead of parsing the graph with rdflib and then perform a SPARQL query.

1.6 Semi-automatic Video Annotation Tool

In this section, we describe the implementation of the annotation tool, all the features available and all the interfaces and results are shown.

1.6.1 Goals and description

The tool allows the manual annotation of the prerequisite relations and concept descriptions in order to build the knowledge graph based on the ontology explained in section 3.4. But, since the manual annotation can be time consuming, we aim to speed up the process by reducing the workload of the annotator.

In fact, to ease the manual annotation, we developed an easy to use interface that does not involve complex notions such as the RDF triples. The annotator is asked only to annotate in real time the video with the times in which there is a concept description and to point out the prerequisite relations. Also, an initial set of concepts are shown to the annotator, this will reduce the amount of time spent in adding concepts, and thus the annotator can focus more on adding the relations. The concepts, which the annotator can easily edit if necessary, are highlighted in the video's transcription, allowing the annotator to see the points in the video where the concept is mentioned. For this reason when a video is firstly uploaded to the system it needs a series of operation that we deepen in section 3.6.2. Once the video is annotated the knowledge graph is created and saved in JSON-LD format on a database.

Moreover, the tool aims also to the semi-automatic or fully-automatic extraction of the knowledge graph, through the use of the burst analysis (explained in section 3.6.5), and in conclusion, it allows several types of analysis on the extracted graphs.

1.6.2 Video Preprocessing

When a video is firstly uploaded to the system a series of operation to process the video are performed, and then stored on the database in order to allow a faster access to the video.

The first step is to download the video and the transcript. This is done using the Python packages *youtube-dl* and *youtube-transcript-api* that returns all the subtitles (both auto-generated and manual-generated) with their starting and ending times.

This will allow to visualize the video synchronized with the transcription and to browse the video through the times in the transcription, and thus, to facilitate the annotation of the prerequisite relations and concept definitions.

Once the subtitles are downloaded, the system will create the transcription. Then, if the subtitles does not have a punctuation, a neural net is implemented in order to add punctuation using *punctuator*. This is a fundamental step in order to be able to do the sentence splitting for the video segmentation.

From the obtained punctuated transcript, the CoNLL is generated using the UDPipe APIs¹⁸, and parsed using the *conllu* library.

In conclusion, in order to facilitate and speed up the annotation process two operation are done: video segmentation and keywords extraction.

¹⁸<https://lindat.mff.cuni.cz/services/udpipe/api-reference.php>

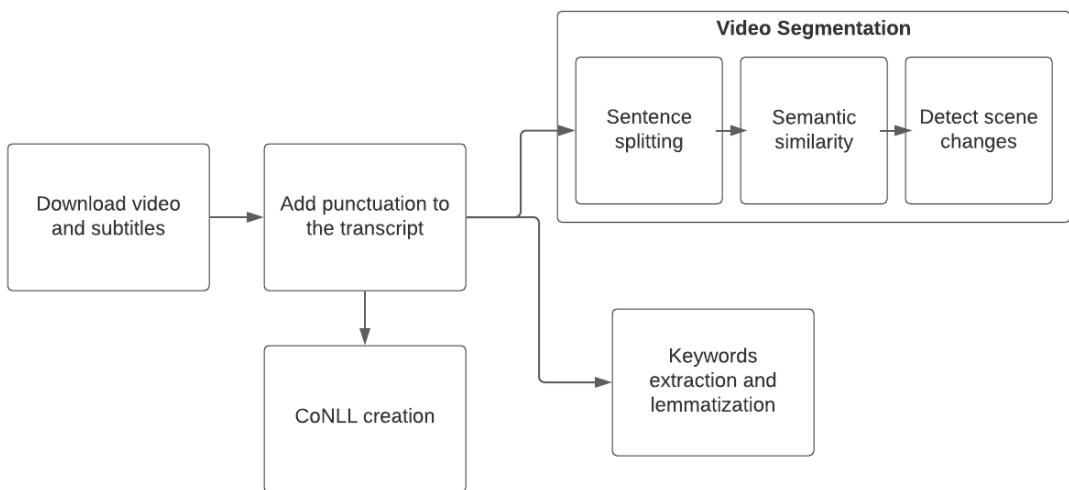


Figure 1.7: Pipeline of video's processing

Video segmentation

To improve usability and reduce time we explored the *NoVoExp* solution [Ver+21] and implemented a very similar approach, which segments the video using semantic similarity, with this approach we aim to partition the video in segments that covers different topics.

For this purpose, the punctuated transcript is divided in sentences using the *nltk* tokenizer, and each sentence is then associated to its starting and ending timestamp, recalculated from the times of the subtitles obtained from Youtube to match the new length of the sentence.

At this point, using *sentence-transformers* we mapped each sentence in an embedding that identify the sentence, converting the sentence in a tensor.

The embeddings are calculated using the BERT model¹⁹, pretrained by sentence-transformer, which can be used for clustering and semantic textual similarity.

For every embedding computed in the previous step, we compute the mean cosine similarity of the current segment, and the cosine similarity between the current embedding and the previous one. If one of the

¹⁹<https://huggingface.co/sentence-transformers/paraphrase-distilroberta-base-v1>

two similarities just computed is above a threshold, we consider the two embeddings semantically similar and thus they will be merged together into a single cluster. If this is not the case, we put the embedding into a new cluster and iterate the process with the remaining embeddings.

At this point we obtain some “raw” segments, and we applied two refinements to improve the results:

- 1) Aggregation of segments shorter than a given duration. In our case, we decided to merge segments until their total length was at least 40 seconds.
- 2) Adjust end and start time of each segment based on detected scene changes.

```

1 while cap.isOpened():
2
3     ret, current_frame = cap.read()
4     cap.set(1, frame_number)
5
6     if ret:
7         current_frame = cv2.resize(current_frame, (240, 180))
8
9         im = cv2.cvtColor(current_frame, cv2.COLOR_RGB2GRAY)
10        h = cv2.calcHist([im], [0], None, [32], [0, 128])
11        h = cv2.normalize(h, h)
12
13        if frame_number > 0:
14
15            diff = 0
16            for i, bin in enumerate(hist):
17                diff += abs(bin[0] - previous_hist[i][0])
18            all_diffs.append(diff)
19            summation += diff
20
21            frame_number += frame_to_skip
22            previous_hist = h
23
24            if cv2.waitKey(1) & 0xFF == ord('q'):
25                break
26

```

```

27     else:
28         break
29
30 cap.release()
31 threshold = S * (summation / frame_number)

```

Code 1.3: Color Histogram - OpenCV

To detect the scene changes we performed a color histogram, by using the OpenCV library, for each frame we compute the distribution of the colors and, if the difference between two frames is beyond a threshold, a scene change is detected. In order to compute the threshold we store all the difference and compute the threshold as a weighted average of all the differences. Moreover, since it's not important to compute the difference for each frame, we take 1 frame at second, in order to improve performances.

Keywords extraction

To facilitate the work for the annotator, it is necessary to present him an initial set of keywords automatically extracted, that he can then edit if necessary. For this reason, we tested a few different approaches using different libraries:

- Gensim ²⁰ which is a python library for natural language processing and information retrieval
- Pke ²¹ is python module for keywords extraction which implements different solutions such as YAKE, MultipartiteRank, PositionRank, TopicRank, TextRank and SingleRank.
- Spacy ²² which is a library implemented in python for natural language processing

²⁰<https://pypi.org/project/gensim/>

²¹<https://github.com/boudinfl/pke>

²²<https://spacy.io/api/top-level>

- keyBERT²³ which is a keyword extraction technique that leverages bert embeddings to create keywords and simple cosine similarity to find the sub-phrases in a document that are the most similar to the document itself. We tested two BERT models.
- Python-rake²⁴ (Rapid Automatic Keyword Extraction) is a python module with the implementation of RAKE.
- PhraseMachine²⁵ which is library that identifies automatically multiwords.

In order to evaluate the correctness of the results, we compared the keywords extracted with the keywords annotated by an expert, and we calculated precision, recall and f1-score. The considered texts are taken from the transcriptions of MOOC Youtube videos, in particular we tested the keywords on five manually annotated videos.

	Precision	Recall	F1-score
Gensim	0.142	0.25	0.158
Spacy	0.07	0.758	0.124
TopicRank	0.311	0.182	0.215
YAKE	0.356	0.226	0.256
SingleRank	0.244	0.142	0.162
PositionRank	0.256	0.139	0.165
TextRank	0.122	0.063	0.076
MultipartiteRank	0.344	0.205	0.239
BERT model 1	0.111	0.063	0.074
BERT model 2	0.167	0.094	0.011
LDA	0.253	0.137	0.164
RAKE	0.4	0.234	0.274
PhraseMachine	0.457	0.186	0.252
RAKE and Phrasemachine	0.459	0.269	0.32

Table 1.1: Keywords extraction results

Results in table 1.1 show that the best method is RAKE.

²³<https://github.com/MaartenGr/KeyBERT>

²⁴<https://github.com/fabianvf/python-rake>

²⁵<https://github.com/slanglab/phrasemachine>

```

1 import RAKE
2 import phrasemachine
3 import spacy
4
5 def rake_phrasemachine(text):
6     Rake = RAKE.Rake(RAKE.SmartStopList())
7
8     rake_words = Rake.run(text, maxWords=3, minFrequency=3)
9
10    # Get the first top 15 results
11    concepts = [j[0] for j in rake_words[0:15]]
12
13    # Load model from spacy
14    nlp = spacy.load("en_core_web_sm")
15    doc = nlp(text.lower())
16
17    t = [token.text for token in doc]
18    p = [token.pos_ for token in doc]
19    ph_words = phrasemachine.get_phrases(tokens=t, postags=p)
20
21    for c in ph_words["counts"].most_common(3):
22        # Only bigrams or unigrams
23        if len(c[0].split(" ")) < 3:
24            concepts.append(c[0])
25
26    # Return lemmatized words
27    return lemmatize(concepts)

```

Code 1.4: Keywords extraction

In order to improve performances and to have more extracted keyword, the proposed approach is to combine RAKE with the three most common keywords extracted with phrasemachine, as in Code 3.4, because they have a very high precision with a final F1-score of 0.32.

The extracted keywords are finally lemmatized using nltk.

1.6.3 Video annotation

Once the video is processed, the video can be annotated through the video annotation interface.

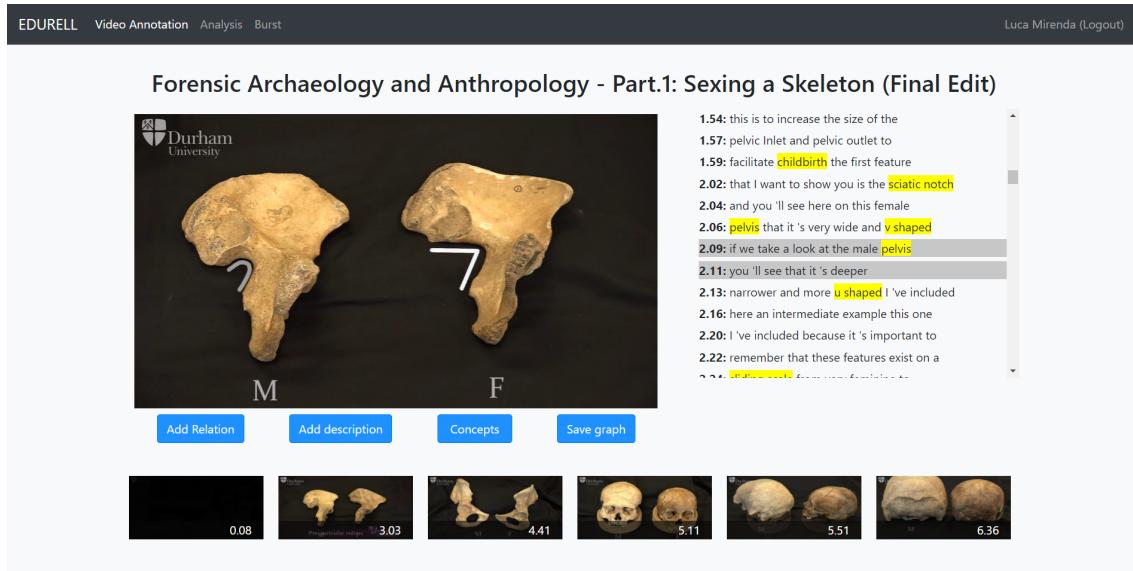


Figure 1.8: Main interface for video annotation

In figure 1.8 there is the interface for the real time annotation. On the left there is the video, and on the right the transcription. The transcription moves synchronized with the video, and it contains the concept highlighted in yellow, in order to make them more evident.

The non-linear consumption of the video is obtained through two methodologies: using the transcript and the segmentation. Therefore, by clicking in one point of the transcript the time of the video will change accordingly to synchronize with the clicked point. Moreover, each segment works as anchor point, each image below the video contains the first frame of the segment, and by clicking on the image the relative segment of the video will start.

Below the video there are four buttons:

- Add relation, it opens the relation interface that allows the insertion of the prerequisite relation.
- Add description, it opens the add description interface that allows

the insertion of the concept description.

- Concepts, it opens the list containing all the concept
- Save graph, it saves the video on MongoDB and it downloads the JSON-LD to the user

Add relation interface

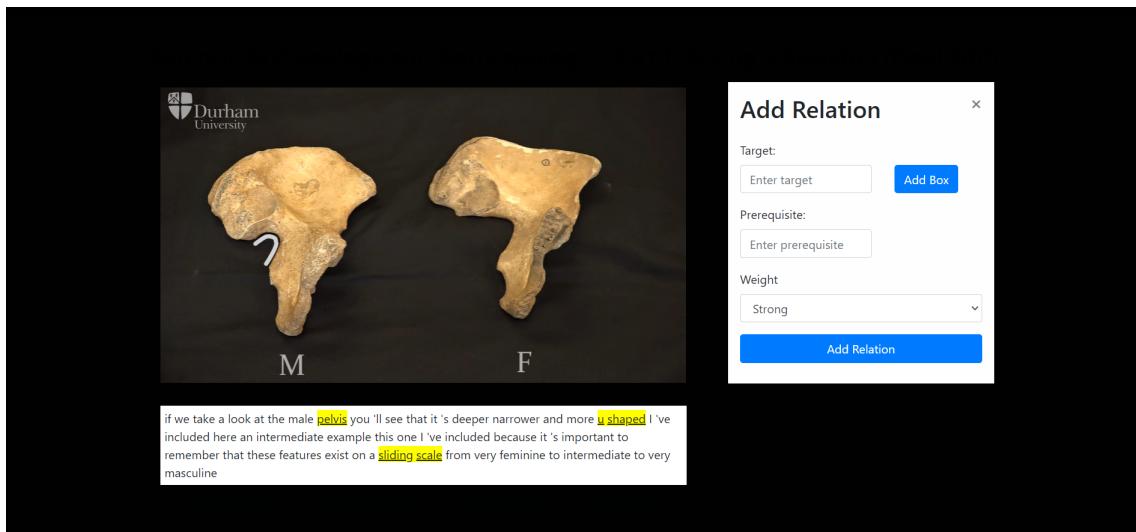


Figure 1.9: Interface for prerequisite relation insertion

In figure 3.9 there is the add prerequisite interface. Here the video is stopped and below the current video's frame there are the last sentences of the video's transcription, with the concepts highlighted.

In order to add the prerequisite relation the user must fill the form with the target of the relation, the prerequisite concept and the weight of the relation.

By clicking on “Add box” it’s possible, but not mandatory, to add the portion of the frame in which there is the concept.

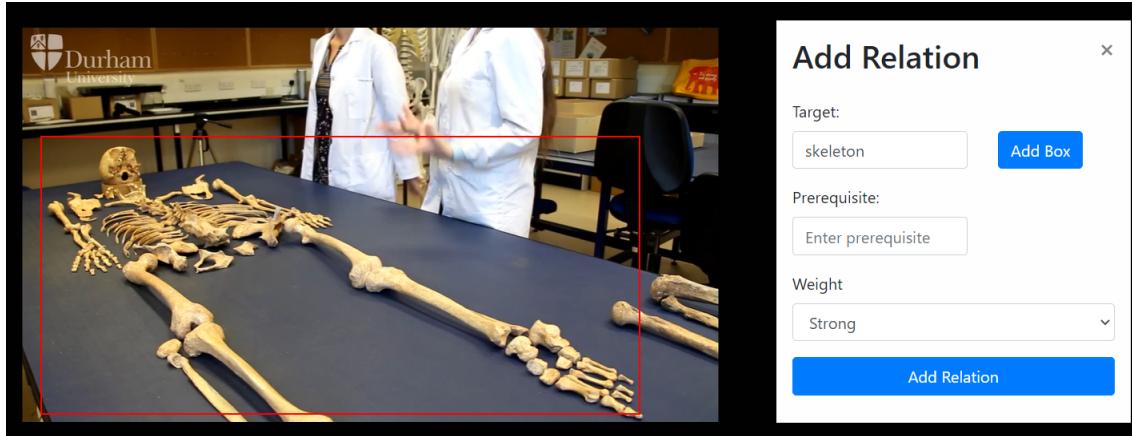


Figure 1.10: Add bounding box

In figure 3.10 there is an example in which the target of the relation is "skeleton" and, since it is present in the video, a box can be added in order to point where the concept is located in the frame.

Add concept description interface

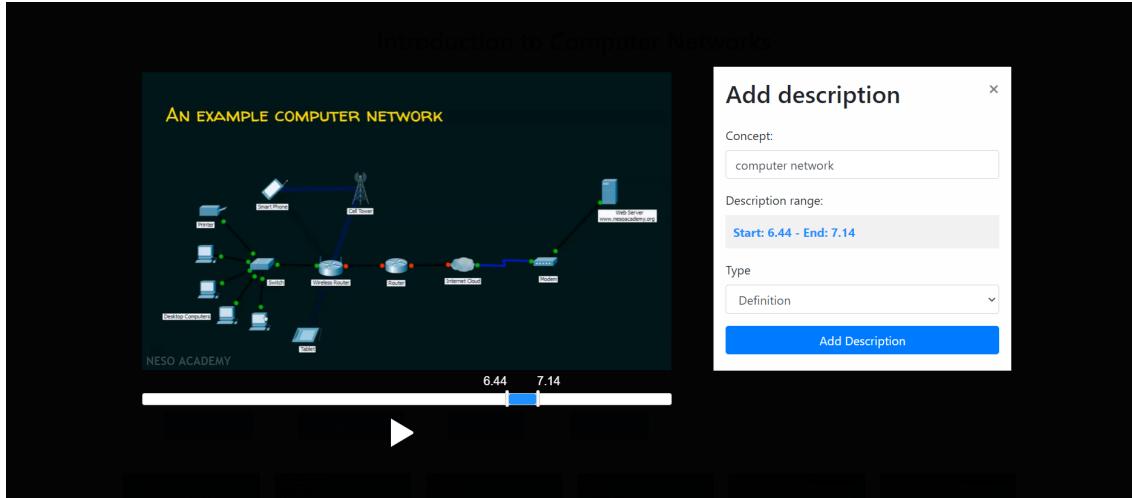


Figure 1.11: Interface for concept description insertion

Here we can insert the concept's description. In order to change the start time and end time of the description, we can slide the bar below the video. The play button is needed in order to re-watch the inserted description.

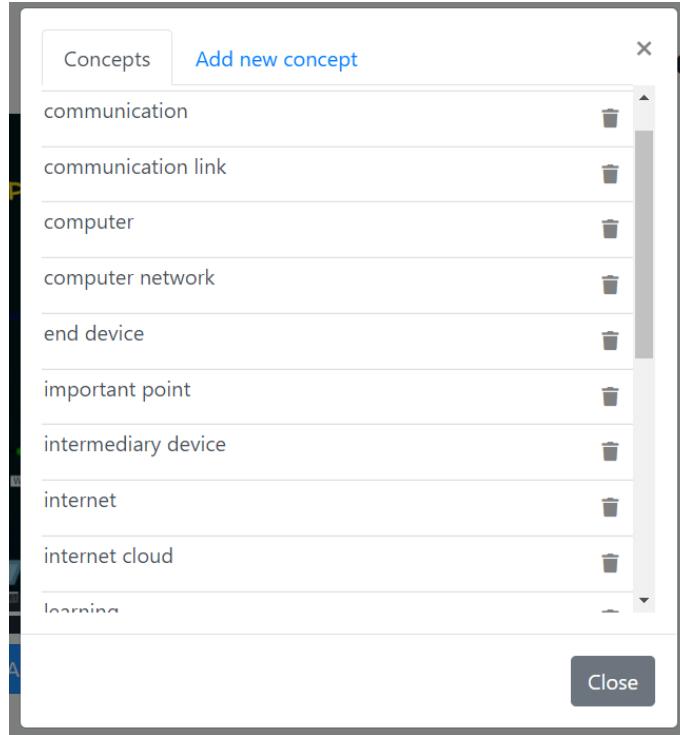


Figure 1.12: Concepts list

Add concept interface

In 1.12 there is the interface with the list of all the concepts, by clicking the trash button it's possible to delete a concept.

Moreover, by accessing to the "Add new concept" tab the user is able to add a new concept through the form in figure 1.13.

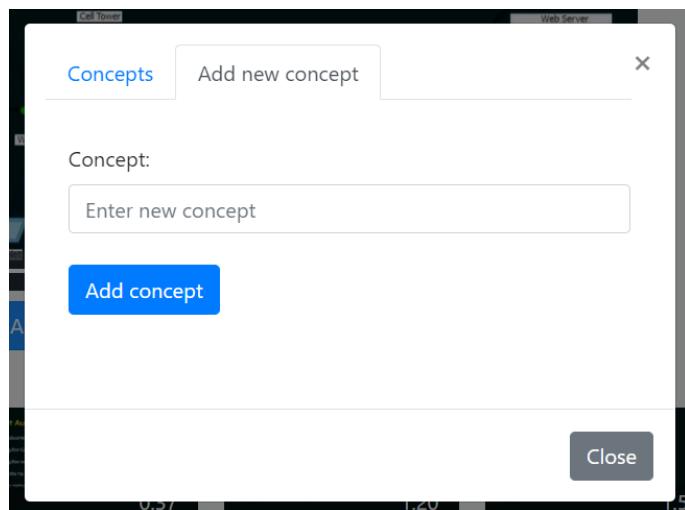


Figure 1.13: Add new concept

The system will check if the inserted concept is in the transcript, and if present, all the occurrences of concept will be highlighted.

Annotation visualization

RELATIONS				Graph	DEFINITIONS			
Target	Prerequisite	Weight	Start time		Concept	Start	End	Type
node	printer	Strong	00:03:34		networking	00:00:46	00:00:57	In depth
communication wired link	wired link	Strong	00:05:01		computer network	00:03:07	00:03:29	Definition
communication wireless link	wireless link	Strong	00:05:42		node	00:03:28	00:03:58	Definition
wireless link	link	Strong	00:06:03		communication link	00:04:23	00:05:07	Definition
wired link	link	Strong	00:06:08		wired link	00:04:54	00:05:07	Definition
internet	intermediary device	Strong	00:06:35		wireless link	00:05:09	00:05:39	Definition

Figure 1.14: Annotation visualization

Below the images of the segmentation there is the interface as in figure 1.14 where it's possible to see all the relations and definitions added. An annotation can be deleted by using the trash button.

By clicking the "Graph" button, the user is able to see the concept map of the video, and thus all the relations inserted. In figure 1.15 there is a simple example of the graph, the concept map says that in order to understand what is a "computer network" the student must first learn what is a "node" and a "communication link". By iterating the process until the root the student can know the learning path.

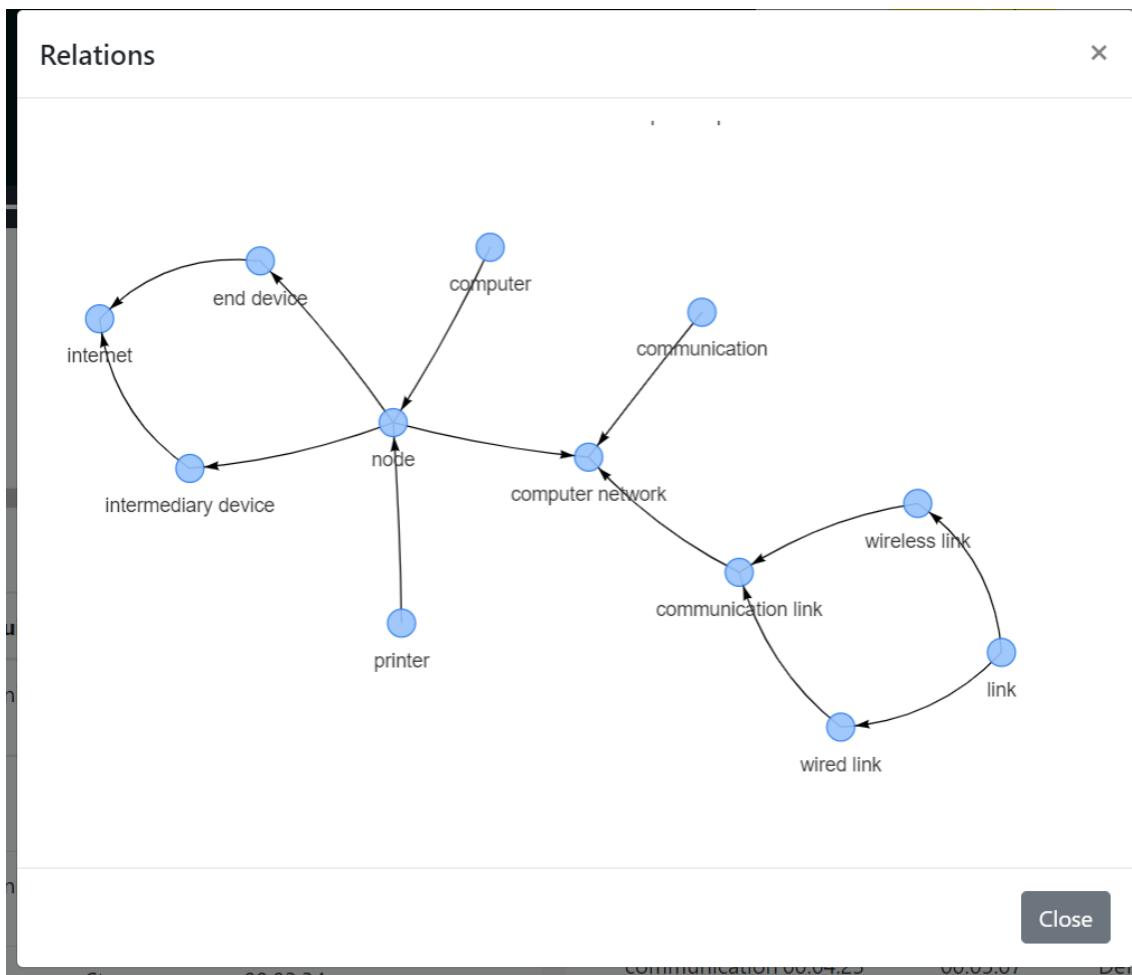


Figure 1.15: Concept map visualization

1.6.4 Analysis

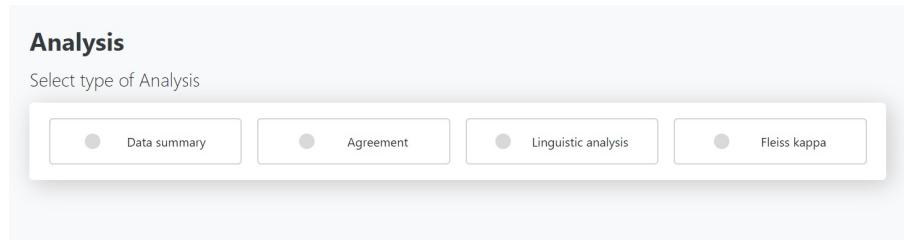


Figure 1.16: Type of analysis selection

The analysis module provides tools to analyze annotations, four types of analysis are implemented:

- Data Summary
- Agreement
- Linguistic Analysis
- Fleiss kappa

Data Summary

Data Summary	
Video title	Forensic Archaeology and Anthropology - Part.1: Sexing a Skeleton (Final Edit)
Annotator	Maria Rossi
Number of total descriptions	37
Number of definitions	27
Number of in depths	10
Number of relations	100
Unique relations	98
Strong relations	100
Weak relations	0
Number of concepts	60
Number of transitive relations	17

Figure 1.17: Data summary

Data summary, as in figure 1.17, shows quantitative analysis of the selected annotation, either considering the graph of relations and concept descriptions.

Quantitative analysis comprehends the number of descriptions (both definitions and in depths), the number of relations (strong and weak), number of concepts and number of transitive relations. A prerequisite relation $A \rightarrow B$ is transitive if it already exists a path from A to B, for instance $A \rightarrow C$ and $C \rightarrow B$.

Linguistic Analysis

Linguistic analysis allows to search for pairs in the annotation that match specific criteria set in the interface (e.g., show all relations where “computer” is prerequisite and the weight is weak) and see them in their context. For each relation it’s also possible to explore their context (next and previous sentence), the linguistic analysis of the sentence where the relation was entered (POS) and the relation graph of the two concepts involved in the relation (Graph).

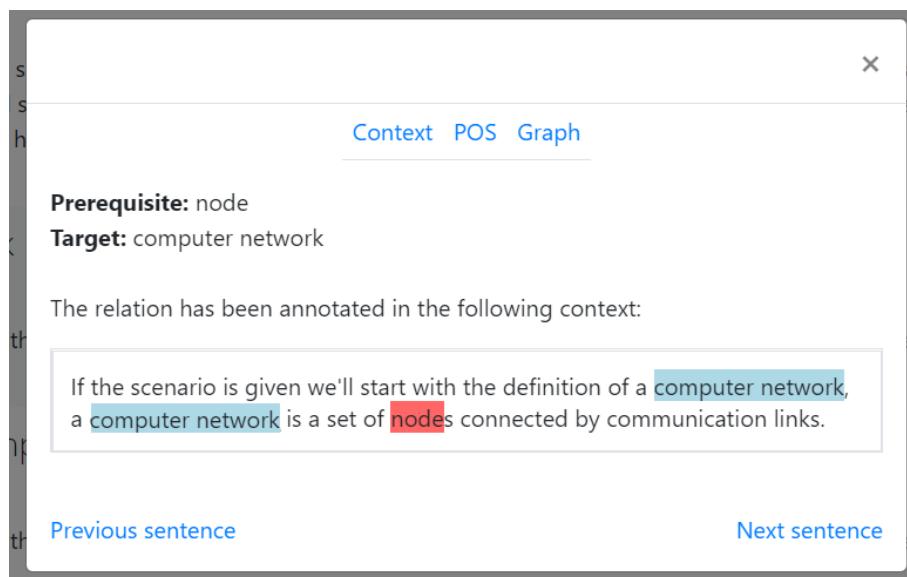


Figure 1.18: Linguistic Analysis

In figure 1.18, we can see an example of the context in which the relation $node \rightarrow computer\ network$ is annotated, showing us the transcript-

tion of the video in the point where the annotator added the relation, and thus why the annotator added this relation.

Agreement

Agreement between two annotations is a measure of similarity and annotation reliability. The agreement is calculated through the Cohen's kappa, which value ranges from 0 to 1, where 1 identifies perfect agreement. Cohen's kappa is defined as:

$$k = \frac{P_o - P_e}{1 - P_e}$$

Where P_o is the percent of concordance between the two concept maps, P_e denotes the agreement expected by chance (i.e., the probability of each individual category). This metric utilizes also the transitivity of relations to check the concordance between two annotations.

Fleiss kappa

EDURELL computes Cohen's and Fleiss' kappa: the first is used between two annotations selected in the analysis interface, the latter is computed between all annotations produced for the same video and is defined in the same way of the Cohen's kappa.

1.6.5 Burst Analysis and temporal reasoning

In order to automatic extract concept maps and concept descriptions we explored the burst [Ado+19] solution, originally intended for prerequisite relations in texts document, that we applied on the transcription of the videos.

Kleinberg uses a two state automaton to describe the periods of an event along a time series (e.g., a document). The automaton is in the first state if the event has a low occurrence, but then it can move to the second state if the event's occurrence it's greater then a threshold, and it can go back to the first state if its occurrence goes below the threshold.

The periods of time in which the event is in the second state are called *burst intervals*.

If applied to the transcription of a video, Kleinberg's algorithm can be used to detect the bursting intervals of concepts. Therefore, we can detect the burst intervals of a list of concepts in order to estimate when the concept are explained. Moreover, we can apply spatial-temporal reasoning on the extracted burst intervals in order to identify prerequisite relations.

Therefore, the algorithm takes as input the text to analyze in CoNLL format and a list of lemmatized concepts, and it's structured in four phases: 1) find words occurrences, 2) extract bursts, 3) detect temporal patterns, and finally 4) extract prerequisite relations.

To find the occurrences of the concepts in the transcription of the video we search each word in the CoNLL file. This allow us to look for all the words of the text with the corresponding lemmatized form.

Then, we extract the bursts intervals, based on Kleinberg algorithm by using the python library pybursts²⁶. These bursts intervals are the points in the video where the instructor is talking about a concept, and they are defined by the starting sentence and ending sentence, that we can easily transform in starting time and ending time. If a concept has more bursts intervals, we take the longest one as "Definition" type and the others as "In depths".

Once the burst intervals are discovered, we can proceed to analyse the temporal patterns by exploiting the Allen's interval algebra. Allen defined all the possible relations between time intervals for temporal reasoning. A relation can be of the types: *X precedes Y*, *X meets Y*, *X overlaps with Y*, *X starts Y*, *X during Y*, *X finishes Y*, *X is equal to Y*. The burst algorithm assigns to each type of relation a weight accordingly to how related is to the prerequisite relations. For instance the relation *X overlaps Y* implies that the concept *X* is firstly explained and its conclusion overlaps with the introduction of *Y*. This is a good indicator of prerequisite relation, and thus this Allen relation has an high weight.

²⁶<https://pypi.org/project/pybursts/>

Finally, for each two distinct concepts, all the weights associated to the burst pairs are combined and normalized. Therefore, the output of the algorithm is every possible prerequisite relation with the weight computed, 0 if there is no relation between the two concept.

The greater the assigned weight is the greater is the probability to have a correct prerequisite relation.

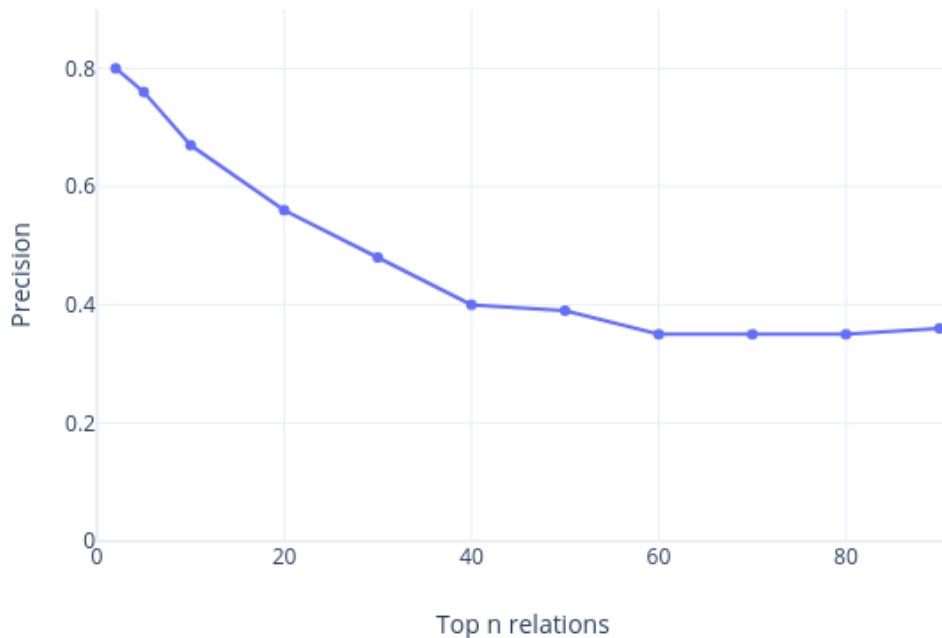


Figure 1.19: Precision of the top relation

As we can see in Figure 1.19, the average precision (obtained by comparing the burst's results with the manual annotations of five videos) decrease as the assigned weight decrease.

Parameters selection

In order to select the best parameters we used a brute force approach by trying all the possible combinations in order to have the best results in the metrics that we describe in next section. The parameters of the burst are the following:

- Threshold: a relation is taken as such, if its weight is bigger than the threshold. For our test we selected as threshold 0.7.
- Top: number of the top relation to take. We took the top 90 relations.
- S and Gamma: they are taken from the implementation of the Kleinberg algorithm from the pybursts library, and they are respectively the base of the exponential distribution that is used for modeling the event frequencies and the coefficient for the transition costs between states. In our case we selected 1.05 and 0.0001.
- Max gap: maximum number of sentences between two bursts after which no relation will be assigned. It is used to reduce the number of 'before' and 'after' relations. We assigned as max gap 1 sentence.
- Allen weights: they are the weight of the Allen relations and they are chosen in compliance of as explained in [Ado+19].

In the following Code 3.5 there is the function to calculate all the metrics with different parameters and to create a CSV file containing all the results obtained.

```

1
2 def get_scores_for_parameters(ann_map, concepts):
3     thresholds = [0, 0.7, 1.5, 2.5, 3.5, 4.5]
4     top_n = [50, 70, 90]
5     ss = [1.05, 1.2, 1.3, 1.4]
6     gammas = [0.001, 0.0001, 0.00001]
7     max_gaps = [1, 2, 3]
8
9     res_df = pd.DataFrame(columns=["threshold", "top",
10                             "s", "gamma", "max_gap", "VEO",
11                             "GED", "Agreement", "LO",
12                             "PN"])
13
14     for t in thresholds:
15         for s_ in ss:
16             for g in gammas:
17                 for m in max_gaps:
18                     for top in top_n:
19                         burst_map = Burst(text, concepts,
20                             video_id, conll, top_n=top,
21                             threshold=t, max_gap=m,
22                             gamma=g, s=s_) \
23                             .launch_burst_analysis()
24
25                         # Calculate all metrics,
26                         # returns a dict with results
27                         res =
28                         calculate_metrics(burst_map, ann_map,
29                             concepts, t, s_, g, m, top)
30
31                         # Append current results
32                         res_df = res_df.append(res,
33                             ignore_index=True)
34
35                         # Write results in a csv file
36                         res.to_csv("results.csv", sep=";", encoding="utf-8")

```

Code 1.5: Burst best parameters

Interfaces for burst extraction

The tool allows the extraction of the graph with the burst method applied with two methodologies: semi-automatic and full automatic.

In the semi automatic approach, the user is asked to insert the concepts that are present in the video. For this purpose, a list of concepts are initially shown to the user, and he can edit as desired. In order to facilitate the insertion, it is also shown the transcript of the video with the concepts highlighted.

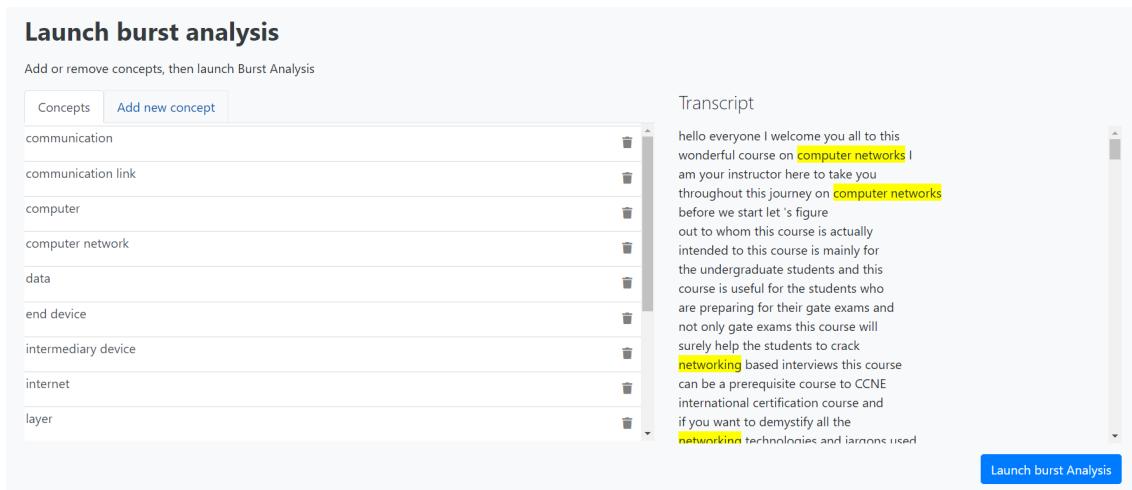


Figure 1.20: Semi automatic interface

The initial concepts that are shown here are the one automatically extracted (as in section 3.6.2), but if a user has already annotated the selected video, the concepts are the one that the annotator inserted. This is useful in order to have a more precise extraction and in order to be able to do a comparison between the burst method and the manual extraction.

In the automatic approach the user is not able to edit the concepts, and the burst analysis is performed instantly.

In both cases, semi-automatic and automatic, once the graph is extracted, an interface containing all the info regarding the extraction is shown to the user. The interface is composed by three tabs: Data Summary, Definition and Relations.

The first tab contains all the quantitative analysis performed by the

Burst analysis results

Data Summary	Definitions	Relations
Video title	Forensic Archaeology and Anthropology - Part.1: Sexing a Skeleton (Final Edit)	
Number of concepts	54	
Number of extracted descriptions	31	
Number of definitions	24	
Number of in depths	7	
Number of extracted relations	90	
Unique relations	90	
Number of transitive relations	25	
Agreement with Maria Rossi	0.406	
VEO with Maria Rossi	0.453	

Figure 1.21: Burst's results interface

data summary analysis, and if a user has annotated the selected video, it shows the metrics in relation with the annotator.

The second tab contains all the descriptions extracted by the burst.

Data Summary	Definitions	Relations																				
	<table border="1"> <thead> <tr> <th>Concept</th> <th>Start</th> <th>End</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>skeleton</td> <td>0:00:08</td> <td>0:01:14</td> <td>Definition</td> </tr> <tr> <td>skeleton</td> <td>0:08:19</td> <td>0:08:56</td> <td>In Depth</td> </tr> <tr> <td>bone</td> <td>0:00:08</td> <td>0:01:39</td> <td>Definition</td> </tr> <tr> <td>establishing sex</td> <td>0:08:19</td> <td>0:09:15</td> <td>Definition</td> </tr> </tbody> </table>	Concept	Start	End	Type	skeleton	0:00:08	0:01:14	Definition	skeleton	0:08:19	0:08:56	In Depth	bone	0:00:08	0:01:39	Definition	establishing sex	0:08:19	0:09:15	Definition	
Concept	Start	End	Type																			
skeleton	0:00:08	0:01:14	Definition																			
skeleton	0:08:19	0:08:56	In Depth																			
bone	0:00:08	0:01:39	Definition																			
establishing sex	0:08:19	0:09:15	Definition																			

Figure 1.22: Burst's extracted definitions

With this interface, in figure 1.22, the user is able to watch the points in the video where the concepts' descriptions are extracted. In the last tab there is a visualization of the relations extracted, and thus the concept map.

In this last tab, it's possible to browse the graph to see the learning paths extracted by the Burst method, as in figure 1.23.

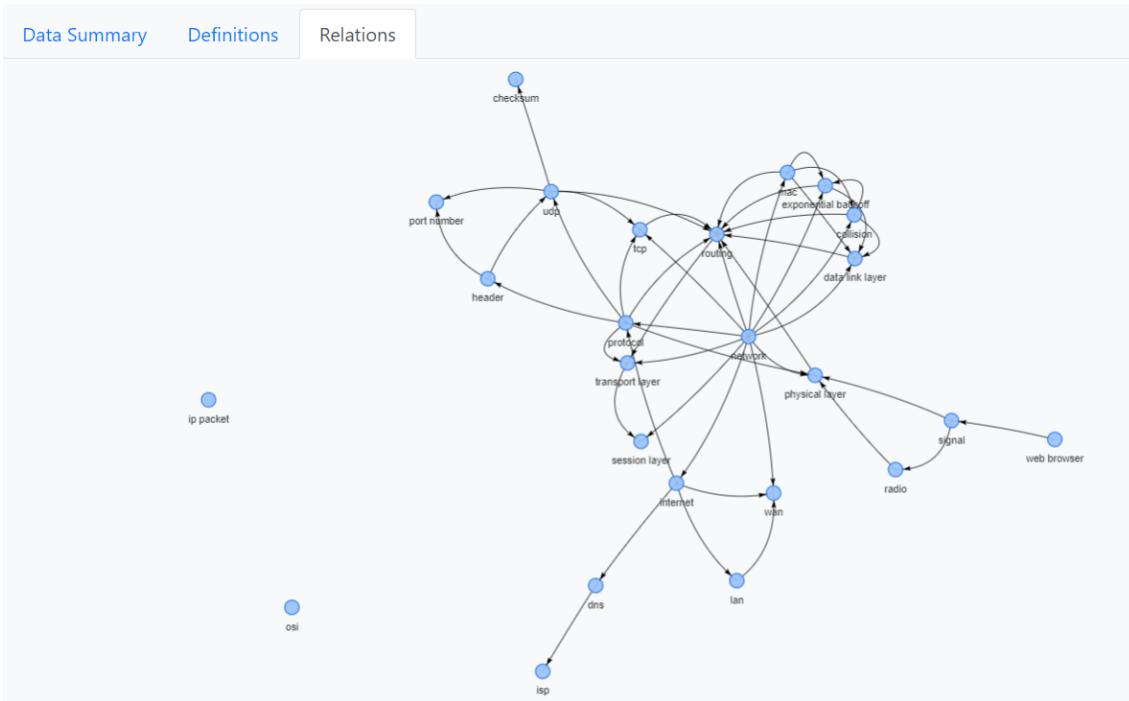


Figure 1.23: Burst's extracted relations

1.6.6 Evaluation

To evaluate the extracted prerequisite relations we compared them with the manual annotated ones. In particular we considered the average of five manual annotated videos and we compared the bursts results with three baseline method's results, using different metrics. All the methods take as input the same concepts which the manual annotator added to his graph.

Baseline methods

Method 1 - hyponyms, hypernyms and meronyms.

A hyponym is a word whose meaning is more specific than its hypernym. The hypernym has a broader meaning than that of a hyponym (e.g. "machine learning" and "supervised learning"). A meronym is in a part-of relationship with its holonym. For example, "finger" is a meronym of "hand". Therefore, prerequisite relationships often exists between hyponym hypernym and meronyms concept pairs. For this reason we extracted from text these relations using the nltk library, exploiting the

Wordnet interface

Method 2 - Reference Distance.

As explained in section 2.3.2 RefD is a metric that models how two concepts refers to each other. We exploited the Wikipedia Python library²⁷ to calculate the reference distance between two concepts, and if above a threshold (which is defined in [Lia+15]) then a prerequisite relation is obtained.

Method 3 - Wikipedia pages.

As third baseline method we used the one proposed by Shuting Wang and Lei Liu in [WL16]. In their approach, they used three sets of features extracted with Wikipedia in order to obtain the prerequisite relations. The first feature is the Usage feature which captures whether a concept is used in another concept's definition. The second feature is the Content Similarity. This feature captures the lexical similarity between Wikipedia concepts using cosine similarity between the concept vector and article vector. The last feature is the Learning Level which measure whether a concept has a lower learning level and should be learned first. To calculate the Learning Level it calculates the range of topic coverage, this says that the more topic the concept cover the more basic the concept is, and moreover it uses the number of in-links/out-links received in Wikipedia pages.

²⁷<https://pypi.org/project/wikipedia/>

Metrics and results on texts

Before the evaluation of the automatic methods on the video's transcription, we tested each method on a chapter of a computer science's book by comparing a manual obtained concept map with the automatic one.

In the RefD method, discussed in [Lia+15], they used the CrowdComp dataset [TC12] to predict whether one page in Wikipedia is prerequisite of another page. With a threshold of 0.05, they obtained an average accuracy of 0.61, but in our results, with the same threshold we obtained an accuracy of 0.33. Moreover, we tested the RefD method with different thresholds and obtained the following F1 score results.

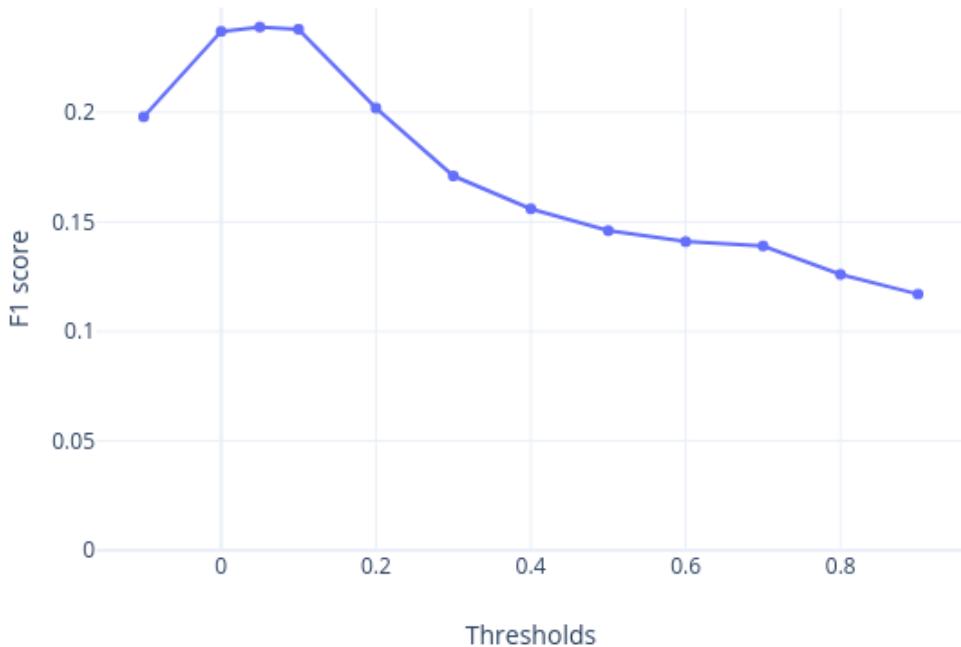


Figure 1.24: RefD results with different thresholds

As shown in figure 1.24 we also got best results with the threshold 0.05 with an F1 score of 0.24. Finally, we tried to get only the top 150 relations with highest Reference Distances and we obtain a precision of 0.356, a recall of 0.034 and F1-score 0.063.

In the Wikipedia Pages method, described in [WL16], they compared

the results obtained with a manual annotation, and by using 60% of the Content Similarity feature and 60% of the Learning Level feature, they obtained an F1 score of 0.52. With the same thresholds we obtained a score of 0.198. Following a table that summarize all the results with the different thresholds, where the columns are the Content Similarity and the rows the Learning Levels.

	40%	60%	80%
40%	0.198	0.198	0.201
60%	0.198	0.198	0.204
80%	0.202	0.204	0.208

Table 1.2: Wikipedia pages results with different thresholds

The Burst method in [Ado+19] for the top 150 relations they obtained an average accuracy of 0.84, in our case we obtained an accuracy of 0.6. Moreover, they obtained a precision of 0.6 for the top 150 relations and we got a precision of 0.7.

Metrics and results on videos

The first metrics that we used to evaluate the performances on videos are the precision and recall, defined as:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Where TP stands for True Positive, which are the number of relations that have been extracted by both the manual annotator and the automatic method, FP (False Positive) are the relations extracted by the automatic method but not by the annotator, and FN (False Negative) are the relations that have been extracted by the annotator but not by the automatic method.

In the following figure we can see the comparison between baseline methods and the burst.

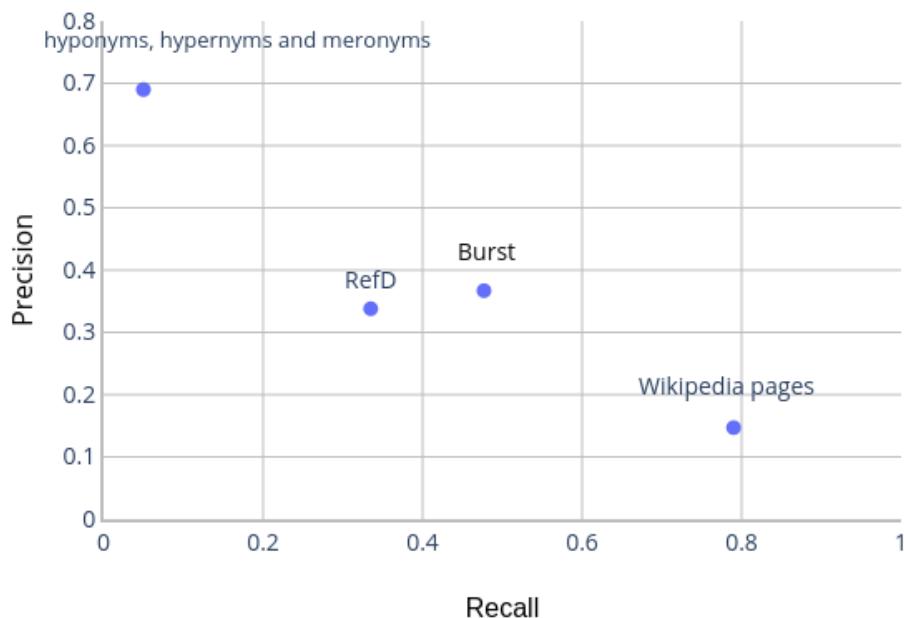


Figure 1.25: Precision and recall of the automatic methods

The results in figure 1.25 show that the Burst method outperforms

RefD in both precision and recall, it has an higher precision compared to the Wikipedia pages, and finally, it has a lower precision than the hyponyms method but a bigger recall.

However, considering the F1-score, which is the harmonic mean of the precision and recall, the Burst method outperforms the others.

The F1-score is defined as:

$$F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Results are shown in table 1.3:

	Hyponyms	RefD	Wikipedia pages	Burst
F1 Score	0.09	0.31	0.25	0.4

Table 1.3: F1 scores prerequisite relations extraction

Vertex Edge Overlap is a basic metric for graph similarity, it measures the similarity between two graphs by calculating the overlap between their edges and vertex. It's defined as:

$$VEO(G, G') = 2 \frac{|V \cap V'| + |E \cap E'|}{|V| + |V'| + |E| + |E'|}$$

Hence, The Vertex Edge Overlap is obtained by dividing the total number of vertexes and edges that are shared between the two graphs, with the total number of vertexes and edges. Following the comparison between the burst method and the baseline method.

	Hyponyms	RefD	Wikipedia pages	Burst
VEO	0.141	0.29	0.322	0.45

Table 1.4: VEO results

The results show that on average the burst analysis produces more similar graphs, and thus, with more vertexes and edges in common with the manual graph.

A well known metric used in graph analysis to measure the similarity between two graphs is the Graph Edit Distance. This metric counts the number of operations (such as edge insertion or deletion) needed in order to transform the first graph into the second one. Therefore, the lower the Graph Edit Distance is, the more similar the two graphs are.

	Hyponyms	RefD	Wikipedia pages	Burst
GED	76.5	146.5	228	97.3

Table 1.5: GED results

Here, results show that the Hyponyms-Hypernyms-Meronyms method has a lower distance, but the burst method outperforms both RefD and Wikipedia pages.

Another metric we used is the Agreement, which we defined in section 3.6.4

	Hyponyms	RefD	Wikipedia pages	Burst
Agreement	0.05	0.13	0.04	0.276

Table 1.6: Agreement results

Agreement shows that the burst concept maps are more in concordance with the manual ones.

Studying the properties of the single concepts can clarify some important features of the concept map. For example, if a concept has an high in-degree, it has a lot of prerequisite and for this reason it is an *advanced* concept. On the contrary, if a concept has an high out-degree, it is the prerequisite of many concepts, and thus it is a *fundamental* concept to learn. In order to evaluate the similarity between the fundamental/advanced concepts of the automatic extraction and the fundamental/advanced concepts of the manual extraction we used the metrics LO and PN.

```

1
2
3 for graphs in graphs:
4     nx_graph = graphs["graph"]
5     rater = graphs["author"]
6
7     leafs = []
8     for x in nx_graph.nodes():
9         if nx_graph.out_degree(x) == 0 \
10            and nx_graph.in_degree(x) >= 1:
11                leafs.append({x: nx_graph.in_degree(x)})
12
13
14     for it in leafs:
15         l_df.loc[next(iter(it)), rater] = it[next(iter(it))]
16
17     roots = []
18     for x in nx_graph.nodes():
19         if nx_graph.in_degree(x) == 0 \
20            and nx_graph.out_degree(x) >= 1:
21                roots.append({x: nx_graph.out_degree(x)})
22
23     for it in roots:
24         r_df.loc[next(iter(it)), rater] = it[next(iter(it))]
25
26 l_df.astype(float)
27 df_l_0 = l_df.fillna(0)
28 LO = df_l_0.corr(method='pearson').loc["Burst", "Annotator"]
29
30 r_df.astype(float)
31 df_r_0 = r_df.fillna(0)
32 PN = df_r_0.corr(method='pearson').loc["Burst", "Annotator"]
33
34 return LO, PN

```

Code 1.6: LO and PN

By using the library networkx²⁸, we create two Networkx graphs, and then we get all the roots and leafs of the two concept maps (manual and

²⁸<https://networkx.org/>

automatic) in order to create a dataframe with as indexes the concepts and as columns the number of in/out degree of the manual annotation and the automatic method. Then we calculate the correlation between the column with the automatic values and the column with the manual values. In order to evaluate the correlation we used the Pearson correlation, as in Code 3.6, which it has results between 1 and -1. 1 for maximum correlation, 0 for no correlation and -1 for negative correlation.

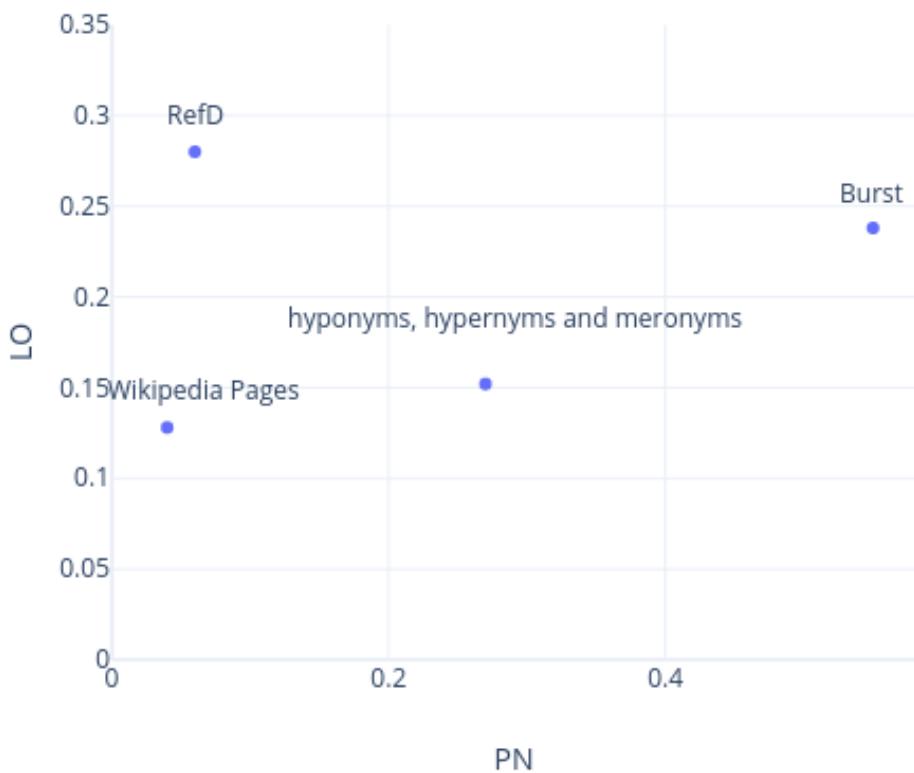


Figure 1.26: LO and PN results

As shown in figure 1.26, all the methods have a low result in the LO metric, but Burst method outperforms the others in the PN metric, showing that the burst method is a good way to find the advanced concepts.

Following in table 1.7 there is summary of all the results obtained and some conclusions about the metrics.

	Hyponyms	RefD	Wikipedia pages	Burst
Precision	0.69	0.338	0.15	0.37
Recall	0.05	0.335	0.79	0.48
F1 Score	0.09	0.31	0.25	0.4
VEO	0.141	0.29	0.322	0.45
GED	76.5	146.5	228	97.3
Agreement	0.05	0.13	0.04	0.276
PN	0.27	0.06	0.04	0.55
LO	0.152	0.28	0.128	0.238

Table 1.7: Summary of the results discussed in the previous pages

The Burst method performs in line with the baseline methods, and in most metrics outperforms the others automatic methods, such as in F1 score, Vertex Edge Overlap and PN metric.

All the results discussed were obtained using five videos from two domains, a computer science domain and an archaeology domain. Following the tables containing the results per domain.

	Hyponyms	RefD	Wikipedia pages	Burst
Precision	0.584	0.363	0.116	0.347
Recall	0.034	0.192	0.733	0.471
F1 Score	0.064	0.241	0.199	0.388
VEO	0.093	0.277	0.27	0.435
GED	102.5	162.0	317.0	119.0
Agreement	0.01	0.079	0.007	0.273
PN	0.057	-0.13	0.125	0.148
LO	0.144	0.292	0.154	0.293

Table 1.8: Summary of the results in the archaeology domain

	Hyponyms	RefD	Wikipedia pages	Burst
Precision	0.8	0.313	0.177	0.387
Recall	0.067	0.479	0.863	0.483
F1 Score	0.121	0.378	0.294	0.428
VEO	0.19	0.317	0.371	0.474
GED	50.5	131.0	139.0	75.5
Agreement	0.099	0.18	0.055	0.28
PN	0.482	0.245	-0.043	0.948
LO	0.161	0.269	0.102	0.182

Table 1.9: Summary of the results in the computer science domain

1.7 Video augmentation for learners

In this section, we describe the implementation of the augmentation tool, all the hypervideo functionalities implemented and all the interfaces are shown.

1.7.1 Goals and description

As already mentioned, educational videos represent one of the most popular formats in online education, but they can also suffer of several limitations. They can for instance be hard to navigate when you need to recall concepts. Moreover, they generally lack of explicit tables of contents or other indexing systems for exploring the video content. With this tool the extracted knowledge graph obtained from the EDURELL Video Annotation Tool will be used to develop augmentation services that help the learner to face some of the issues above. In particular, the proposed application offers anchor points for the concepts that the lecturer is explaining. Additionally, it shows the prerequisites that the student has to know in order to get the current concept, with an interactive concept map.

1.7.2 Video Exploration

When the user log to the system it accesses to the dashboard with the following interface in figure 1.27.

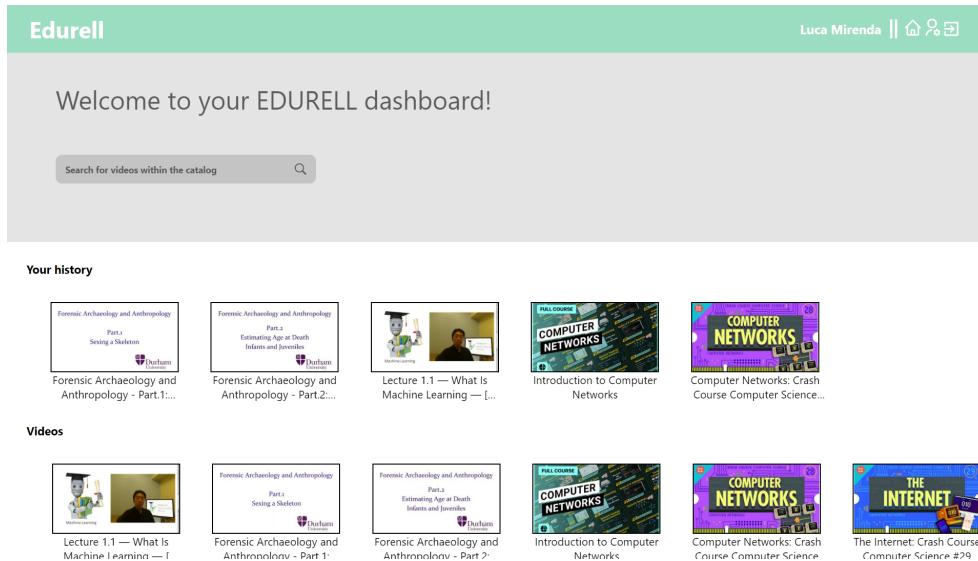


Figure 1.27: Video augmentation tool dashboard

Here it's possible to look for a specific video through the search bar, or to continue to watch a previous watched video through the history section. When the user enters in a video it accesses to the following interface for the video augmentation.

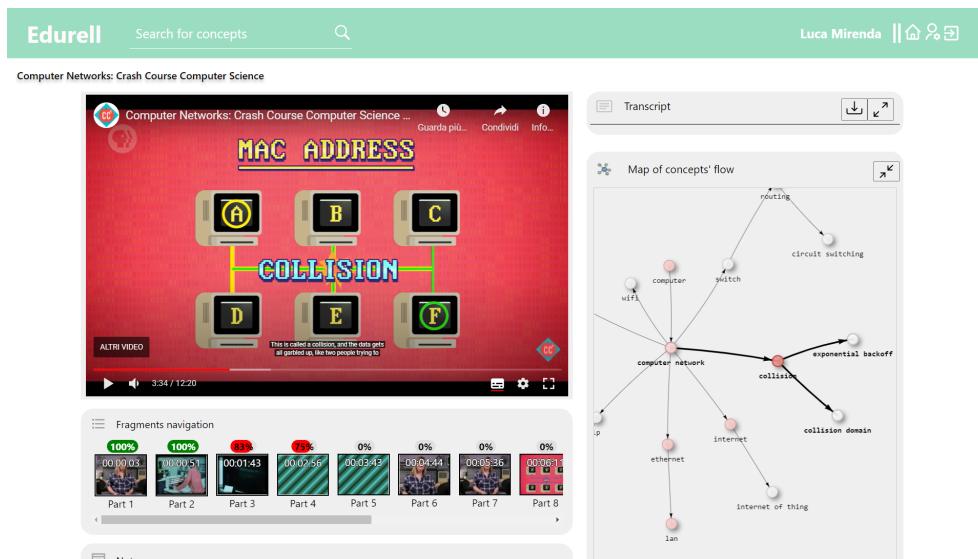


Figure 1.28: Video Augmentation

On the right of the video there is the *Transcript* container and the *Map of concepts' flow*. The user can enlarge or reduce both containers. The transcript container shows all the subtitles synchronized with the video, and by clicking on a subtitle the video will redirect to the selected time. These subtitles are obtained through the youtube-transcript-api and they can be downloaded with the corresponding button at the top of the transcript.

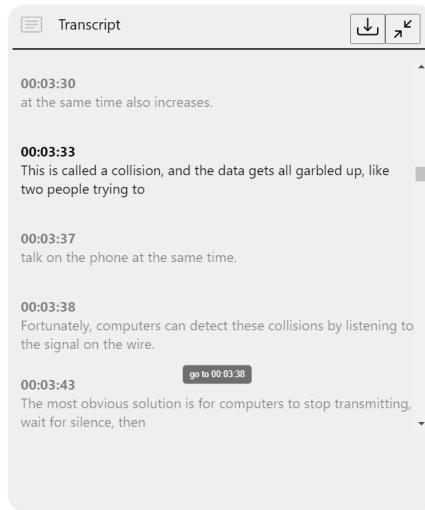


Figure 1.29: Transcript

The Map of concepts' flow container shows the concept map of the video, as in figure 1.30.

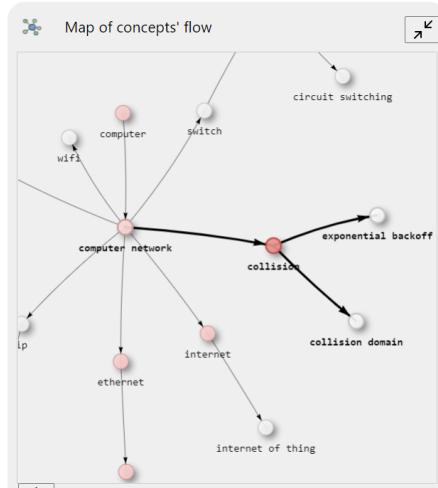


Figure 1.30: Map of concepts' flow

This panel allows you to take advantage of the prerequisite relations to browse the video and deepen some concepts. When a prerequisite relation starts, the involved nodes are colored with light red, moreover, the system will automatically zoom the map to the concept that is being explained at the current moment, for instance in figure 1.30 the concept *collision* is being explained and for this reason it's highlighted to the user with the color red. In this way the user is able to see which concepts are needed, and by clicking on a concept, markers appears in the video's bar to highlight where the concept is explained. This allows an easy video navigation in order to reach all the concepts needed in a faster way.



Green circles are the main definitions of the concept

Additionally, by searching a concept in the search bar a subgraph of the concept is displayed to the user.

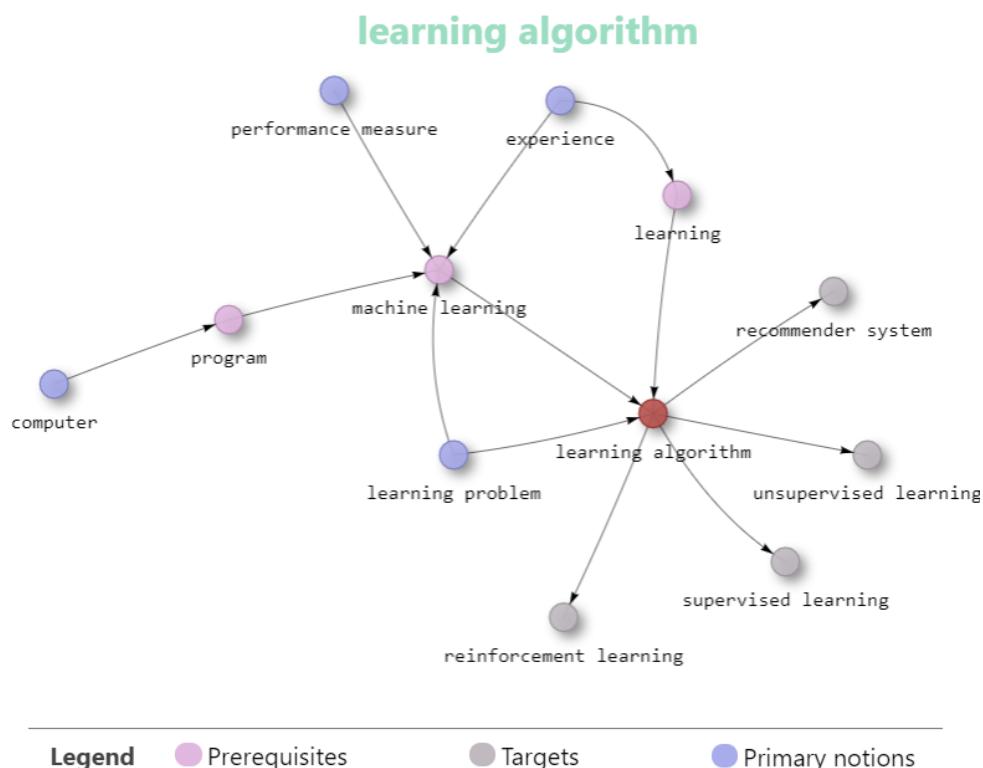


Figure 1.31: Subgraph of a concept

This subgraph shows to the user the learning path of the concept. In this visualization the red concept is the one that the user selected, the blue concepts are the *primary notions*, which are the fundamental concept that does not have any prerequisite, the pink concepts are the prerequisites of the selected concept and the grey concepts are the concept that need the selected concept in order to be learnt.

Moreover, below the video there is the segmentation for the non linear consumption of the video as in figure 1.32.

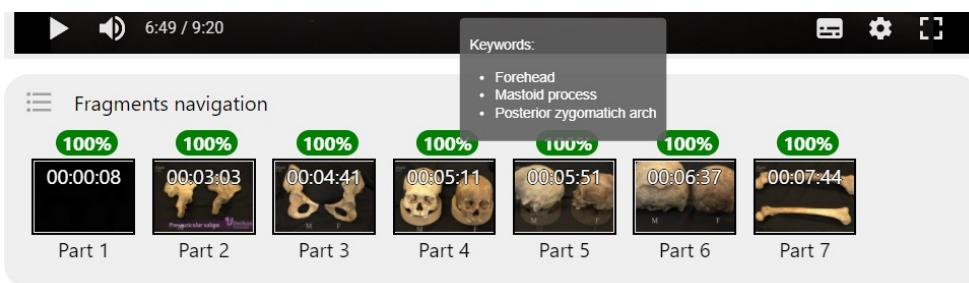


Figure 1.32: Non linear consumption

Here, by hovering the mouse on an image, it shows all the concepts that are explained in that part of the video, thus a student is able to skip to the point he need.

The last feature implemented is the ability to take notes while watching the video.

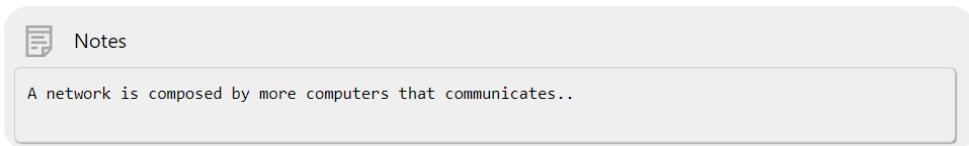


Figure 1.33: Notes

Figure 1.33 shows the textboard in which a student can type his notes about the topic he is learning.

1.8 Server deployment

In order to deploy the project and to allow the access at all the annotators and learners, we used Gunicorn²⁹ which is a Python Web Server Gateway Interface HTTP Server for UNIX. Gunicorn allows the communication between the web server and an application.

Nginx³⁰ is a light with high performances web server, which takes care of handling https connections and the requests that are meant to reach our applications and pass them through Gunicorn. So, Gunicorn works as in between web server and the Flask application.

The server must support also the deployment of both the Annotation tool and video augmentation tool. In order to allow the simultaneously run of the two projects, Nginx will redirect the HTTP request to two different locations.

```
server {
    listen 130.251.47.105:5000 default_server;

    location / {
        root /var/www/se21-p21/src/react-app/build;
        index index.html;
        try_files $uri $uri/ =404;
    }

    location /api {
        include proxy_params;
        proxy_pass http://localhost:5000;
    }

    location ^~ /annotator/ {
        include proxy_params;
        proxy_pass http://127.0.0.1:5050/;
    }
}
```

Figure 1.34: Nginx configuration

²⁹<https://gunicorn.org/>

³⁰<https://www.nginx.com/>

The annotation tool is implemented using the location "annotator", in this configuration every request to "*/annotator*" will be redirected to the annotation tool and the requests to "/" will be redirected to the video augmentation tool.

In conclusion, in order to allow the automatic start of the tool we used a service that starts automatically on server boot.