QA Classifier
Team 11
Larissa Djoufack Basso, Rachell Kim, Weiming Long
Design HW2

The Internet has become irreplaceable in the field of education that it grants access to everyone with its immense resource. It is simple for educators to record videos of different fields of study and post them on their YouTube channels as well as other educational videos website. Both teachers and students can benefit from these videos for research purposes and individual study. Websites like YouTube, Khan Academy, MIT OpenCourseWare, and many other educational websites grant everyone easy access to study from millions of educational videos. Several meta-analyses have shown that technology can enhance learning, and multiple studies have shown that videos, specifically, can be a highly effective educational tool. However, with this exponential growth in the number of educational videos, one can find it overwhelming to find a suitable and effective video to study from. The name of our project is called QA Classifier, and our purpose is to make a study using online educational video resources.

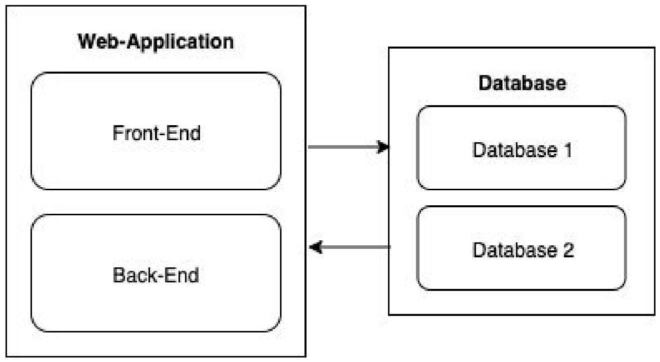
As a team of college students, we all have been watching videos to tackle difficult problems during our study. Since there are so many educational video resources we can access, it sometimes seems inefficient and time-consuming to find the video that we need. It seems almost impossible for us to choose from so many educational videos websites and so many video websites. A good classification of different topics should be added. The other thing that's always annoying to us is that many educational videos can be as long as one and a half hours or more. Traversing through the video to find some specific key points is just challenging as finding needles in a haystack. Forwarding the video too fast can easily skip the key points we are looking for while forwarding slowly will be too time-consuming and exhausting. These problems of using current online educational video resources really lowered our productivity and this is the main reason why we want to bring new changes.

In order to use those online resources in a better way, we plan to build a QA classifier that can analyze educational videos and make specific knowledge points much easier to find. Our system will bring great benefits to students and those who want to extend their knowledge. Finding the right educational video and even specific part in a video will make online learning much more efficient and effective than ever before. This system can go even further into different fields to extract and classify information from videos. Our system will benefit not only students, but also every person in society who wishes to seek knowledge.

The QA Classifier is a web-application that parses and classifies segments of recorded instructional videos. With this system, users may search for spoken keywords and topics within uploaded videos using a search bar from our web-interface. Once an instructional video is uploaded into the application, the system will then parse its contents and automatically tag relevant keywords to certain time frames. The video, the keywords, and time frames associated with the keywords within the video are then stored into the database, later to be retrieved when users query for the tags associated with that video.

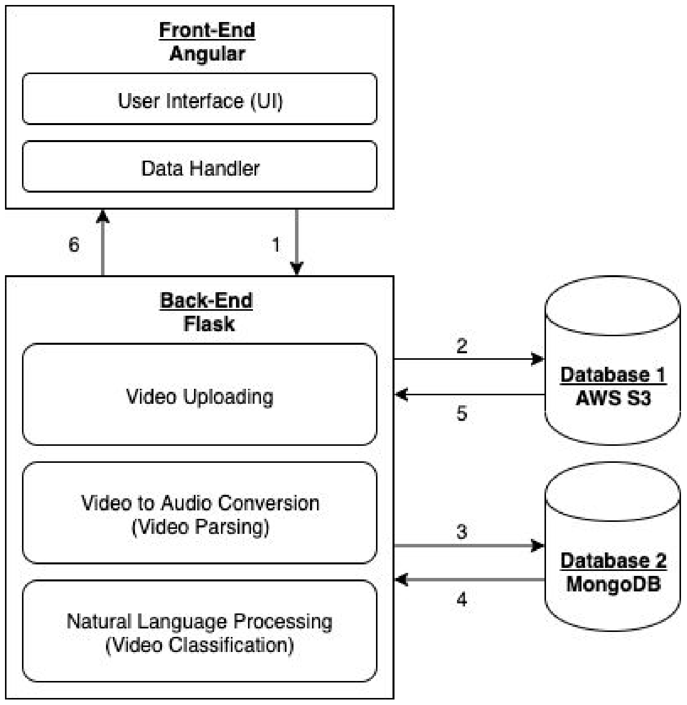
Like most web-based systems, the QA Classifier may be divided into two parts: the database and the web-application. The database container may be further divided into two different types of databases, one for storing the actual videos themselves and one for storing the metadata associated with them. Our team chose to use the Amazon S3 Standard Storage to store the video contents and MongoDB, a NoSQL database, to store metadata about each video. The web-application container may also be divided into two parts consisting of the front-end application and the back-end application, managed by Angular and Flask respectively. This division between the front and back-end applications allows our system to mediate the flow of data between the users and the databases. Figure 1 shows the generalized system architecture of the QA Classifier.

Figure 1: System Architecture I



Within each container, a number of components are run or supported. The web-application container, for example, has video parsing, video classifying, and video uploading components in the back-end, while the front-end has components to support the user interface (UI) and manage user input data. The database container provides the infrastructure to support the components within the back-end application. The details of the components and their associated containers are laid out in Figure 2.

Figure 2: System Architecture II

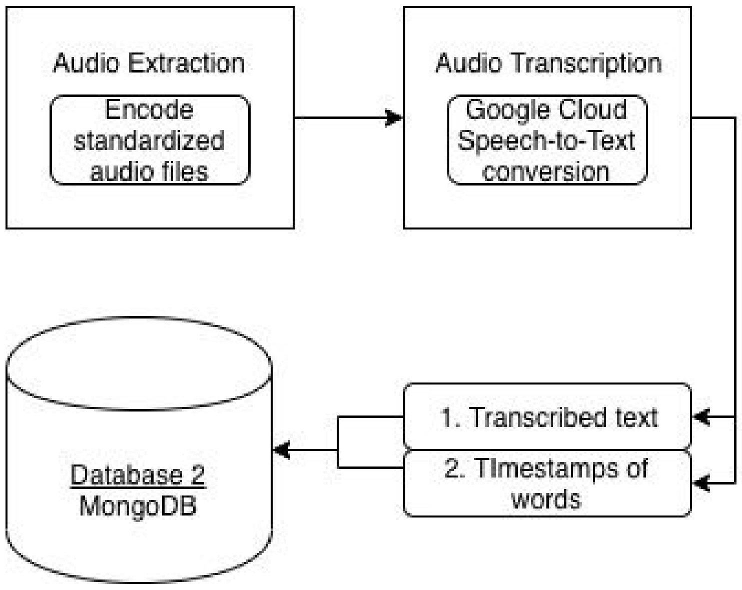


The overall workflow of our system and the interaction between the containers' components can be understood through the upload and search processes. First, the user uploads a video through the UI managed by Angular, the front-end application. The video data is then routed to the back end of the application, run by Flask, via the data handler within Angular, and follows the arrow labeled "1" within Figure 2. Now in the back-end portion of the application, the video data is uploaded into the Amazon S3 database, labeled with arrow "2". Additionally, the video itself is passed onto the video parsing section of the back-end, where the audio of the video is extracted and converted to text data. The text data is then passed onto the video classification section, where natural language processing is performed to extract key data from the full text. The keywords extracted from this text file, along with the time frames associated with them, and the link to the original video in the S3 database is then stored into MongoDB, following arrow "3". This completes the upload, parsing, and classifying process. Now, if the users query for certain keywords using the search bar within the UI, the search words will be sent to the back-end via arrow "1", and queried within MongoDB. Before sending the query data to the back-end, search queries made by the users must first be validated, parsed, and packaged appropriately in JSON format. This process is taken care of by the data handler component within the front-end application. Then, when the query data is sent to the back-end application, all search words are queried in MongoDB. Once the appropriate videos containing the search words are found, the time frames and associated keywords are returned to the user through arrow "4". The video itself is also retrieved from the S3 database through arrow "5". Both the video and the video time frames are sent to the front-end, following arrow "6", and put in a list format within the UI. The relevant videos are now available for viewing to the users, and this completes the video search portion of our application.

During the video parsing process, our system extracts audio files from videos and then transcribes them into useful text files for later use. Our system uses an API from FFmpeg to handle the audio extraction and formatting. The video files uploaded into our system can vary in format and encoding, and thus we incorporated the FFmpeg API to handles these discrepancies. This API also allows us to output the audio files in a way such that the format is standardized (i.e. having the same encoding, sample rate, bits per sample, and the number of audio channels). The standardized format we are using is 44.1 kHz with dual audio channels encoded in FLAC. To ensure the quality and accuracy of the audio transcription, we are setting the output audio files to have high fidelity. Once these standardized audio files are produced, our system then takes them to the next step, which is audio transcribing through the Google Cloud API. We first need to upload our audio file to the Google Cloud Storage (GCloud) first before transcription while the video file itself is being stored in the Amazon S3 database. Because the GCloud only allows local file transcription for audio files less than 10 Megabytes, we must first store our audio files in the GCloud storage. This will allow us to transcribe audio files larger than 10 Megabytes through this API. The GCloud Storage system is not a complicated database, but instead a bucket of files. It is fairly easy to access the file with a URL link which contains the bucket name and the file name. Our system uses the Speech-to-Text API from Google Cloud, currently one of the most powerful neural models on the market, to transcribe the audio to text. The features within the API to accurately transcribe proper nouns and mark time offsets of words were deemed to significantly benefit our application. With the encoding-standardized audio files obtained from using FFmpeg, our system is able to transcribe smoothly and output text files. There will be two files produced as a result of this process: one is the transcribed text of the whole audio and the other one is a list of every word and its time offset with respect to the original video. These two files will be used in the later stages of classification and retrieving the time frames of keywords.

In order to let the video parsing portion run automatically when the videos are sent to the back-end, we are using a Python script to run our required Python files and the OS library in Python to call the system command-line operations. The FFmpeg will be run in the system command line as the video file arrived. After the audio file is converted, the system calls a Python file that uploads it into our Gcloud storage bucket and gets its URL link back. With the URL link, the system then calls another Python file that performs Gcloud Speech-to-Text Transcription and passes the URL as the argument. When the audio transcription is finished, the two output files will be stored in the MongoDB for classification purposes.

Figure 3: Video Parsing Flow Chart

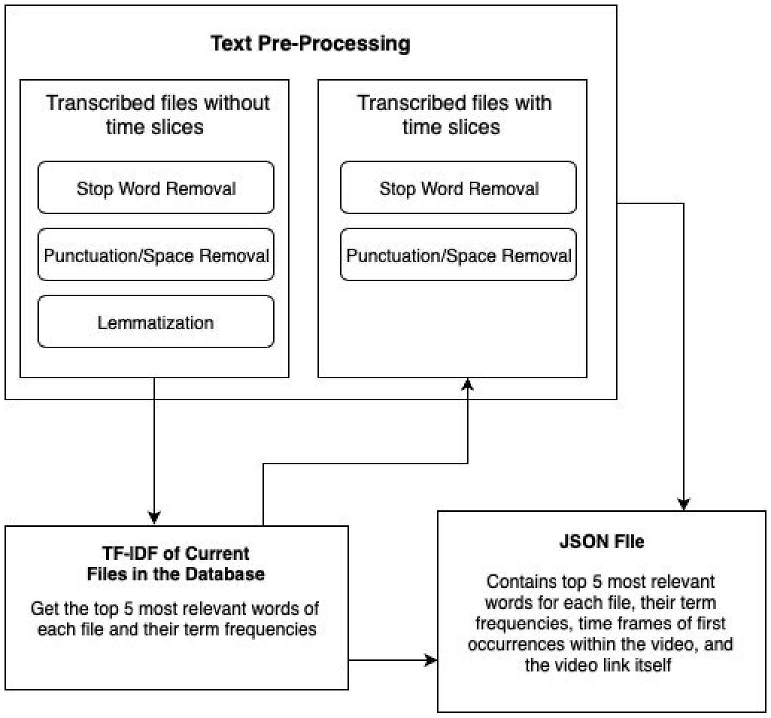


The video classification portion of the project will perform an analysis of the transcribed files obtained from the video processing and will implement machine learning algorithms such as Term Frequency-Inverse Document Frequency (TF-IDF) and topic modeling. As mentioned above, two transcribed files will be generated per video. The first transcribed file will provide a plain transcription of the content of the video without any time slice information for each word. The second transcribed file will provide time slices of all words spoken in the video including the time slices in the video in which those words occur. This section of the project first pre-processes transcribed files by removing stop words, punctuation, white spaces, then performs a lemmatization on each of the words in the text files.

The next step after text pre-processing is the implementation of TF-IDF. TF-IDF provides the relevancy of a word among a set of documents. Therefore, words present in high frequency among all documents will receive a lower relevancy weight compared to high-frequency terms unique to a document or file. In the QA Classifier, we will use the top five words with the highest frequencies per files as video labels. Those words will be stored in a JSON format along with the link to the video they appear in, the first time occurrence (timeslices) of those words in videos and their weight frequency. We also decided to use a python dictionary to store previously defined data on each video because it stores data like a map, are indexed, has key values and is changeable. Those are important for our project because it enables us to easily store and retrieve data on videos in the MongoDB database, in a JSON format. In the front-end, of our system, underneath each video, we will provide 5 labels, which consist of the most relevant five words of that specific video. By clicking on those labels, the video will start playing at the section of the video where the word first occurred.

In cases when the user searches for keywords not present in our generated labels, in the back-end, we implement topic modeling using GenSim on each file and determine the topic containing the greatest number of keywords the user searched for. Later, we select labels that fall within those topics and return the most relevant labels to the front-end as part of our search result. Topic modeling is an unsupervised clustering matching algorithm which matches similar words within a file together. Therefore, words that fall under the same cluster or community belong to the same topic.

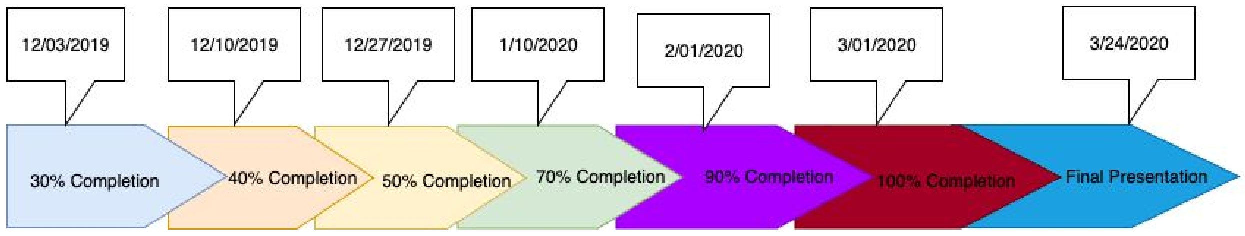
Figure 4: Video Classification Flow Chart


Text Pre-Processing
Transcribed files without time slices
Stop Word Removal
Punctuation/Space Removal
Lemmatization
Transcribed files with time slices
Stop Word Removal
Punctuation/Space Removal
TF-IDF of Current Files in the Database
Get the top 5 most relevant words of each file and their term frequencies
JSON FIle
Contains top 5 most relevant words for each file, their term frequencies, time frames of first occurrences within the video, and the video link itself

In our system, the pattern architecture we use is the layered architecture. As shown and described above, the developers working on each component or layer only needs to know about the layer above and below and each component within the system performs a specific role within our system (i.e video classification, video parsing, front-end). In our system, the output of one component is the input to another. The implementation of layer architecture in our system provides great advantages. This architecture makes it easy to design, develop, test, and assign responsibilities among different members of our team. Moreover, significant changes made to one component do not necessarily impact the other ones.

In the planning process of our project, we came up with milestones or different phases for our project implementation with specific end dates as shown in the milestones diagram below. Each phase is represented by a unique color, and the deadline for each phase is shown. Each phase is comprised of development, testing, integration and documentation. Initially, each team member worked on his or her individually assigned component. These components were integrated into one module to meet the 30% completion deadline. Once all components were integrated into the main application, our team began the 30% completion testing phase. The 30% demonstration will showcase communication and interaction of the different portions of our system. That is, we will generate our own dataset of videos and audios on different topics, which will then be passed on and transcribed into files using the program created by one of our team members. The transcribed files will be processed by the natural language processing portion and generate 5 tags per video, which will then be stored into a JSON format along with term frequencies. The tags as well as the link to videos will be stored in our MongoDB database. This will allow the users to query real data processed by our system from the web-interface. Thus, majority of the work put into the 30% demonstration will involve transferring individual code into the back-end section of the application framework. The 40% completion milestone will involve refining the 30% version of our application. For instance, our ability to obtain precise data, good quality topic modeling, or valid tags heavily depends on the quality of text pre-processing. In the 40% demonstration, we will showcase a refinement of our text pre-processing by either implementing lemmatization instead of stemming or a combination of both stemming or lemmatization. The 50% will be a refined version of the 40% version, and so on and so forth. While the earlier versions of our application will mainly involve integration and focus on smooth communication between containers and components, the latter half — the 50%, 70%, 90%, and 100% — will mostly involve rigorous testing and optimizations for our system. Some optimizations to consider for the final stages of development include speeding up the upload process, speeding up the transcription process, creating a more friendly and aesthetically pleasing UI, and allowing a login system to store user information. A week prior to the end of each phase, we test all components integrated together with various data. Also, we use GitHub and Trello to keep track and update the projects we work on. On Github and Google Drive, we have documented installation steps and lists of required libraries to run specific components. Thus, all stages of development include thorough documentation of the changes of our evolving system.

Figure 5: Milestones



<u>**Component assignments:**</u>

1. Larissa
   a. Natural Language Processing (Video Classification)
2. Rachell
   a. User Interface
   b. Data Handler
3. Weiming
   a. Video to Audio Conversion (Video Parsing)