QA Classifier
Team 11
Larissa Djoufack Basso, Rachell Kim, Weiming Long
Design HW1

Our system, 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 it's 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 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. Figure 1 shows the generalized system architecture of the QA Classifier.

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 front-end and back-end web-applications. The details of the components and their associated containers are laid out in Figure 2. 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. 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). Once these standardized audio files are produced, our system takes them into the next step of audio transcribing. The 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.

The video classification portion of the project will perform an analysis on the transcribed files obtained from the video processing and will implement machine learning algorithms such as Term Frequency-Inverse Document Frequency (TF-IDF). 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. This section of the project first pre-processes transcribed files by removing stop words, punctuation, white spaces, and by performing a lemmatization of words. 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 a high frequency terms unique to a document or file. In the QA Classifier, we will use the top five words with the highest frequency 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. The JSON file will be stored in the MongoDB database which will be parsed and uploaded to our website. 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 user typed-in keywords. Later, we select the labels that fall within that topic. Topic modeling is an unsupervised clustered matching which matches similar words within a file together. Therefore, words that fall under the same cluster are under the same topic.

Component assignments:

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)

Figure 1: System Architecture I

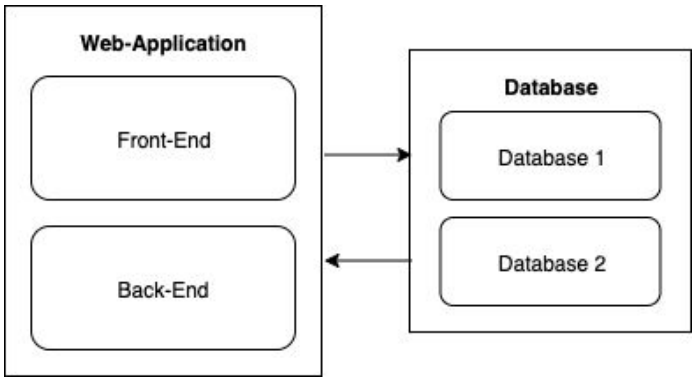

Figure 2: System Architecture II

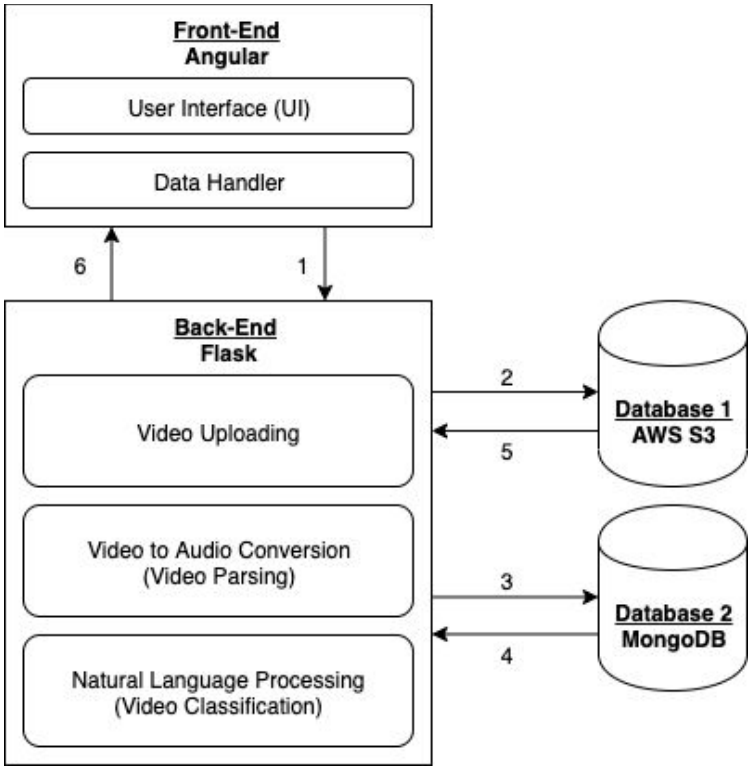

Figure 3: Video Parsing Flow Chart

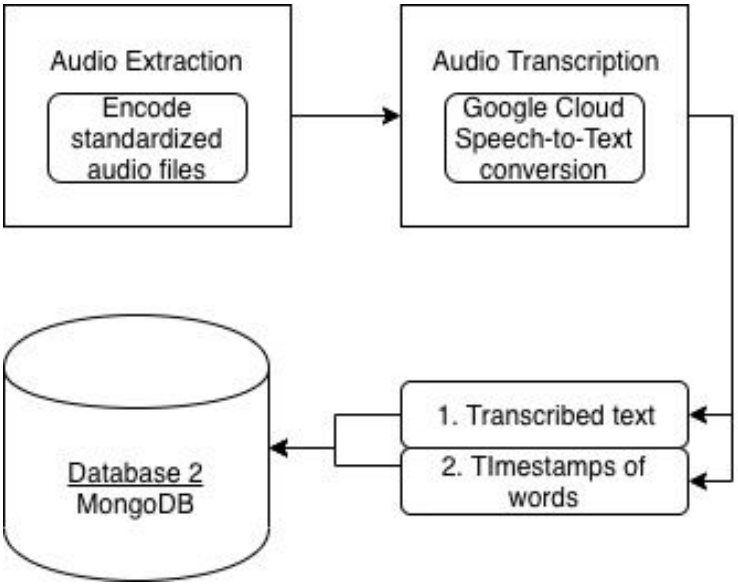

Figure 4: Video Classification Flow Chart