

# DigiClips Media Search Engine

DESIGN DOCUMENT

**Team:** sddec21-06

**Client:** DigiClips Inc.

**Adviser:** Dr. Ashfaq Khokhar

**Team Members:** Tyler Johnson, Samuel Massey, Max Van de  
Wille, Maxwell Wilson

**Email:** sddec21-06@iastate.edu

**Website:** sddec21-06.sd.ece.iastate.edu

# Executive Summary

## Development Standards & Practices Used

In this project we will not be designing or implementing circuits or hardware, therefore all standards and practices will be software focused. Specifically, for this project we will be using practices such as object-oriented programming as well as the following IEEE standards for software development best practices:

- IEEE Std 1063, Standard for Software User Documentation
- IEEE Std 829 –2008, Standard for Software Test Documentation
- IEEE Std 830-1998, Recommended Practice for Software Requirements Specifications
- IEEE Std 1012, Standard for Software Verification and Validation

## Summary of Requirements

- Speech-to-text for television and radio recordings
  - Process recording files to extract text from audio.
  - Store text in searchable keyword/phrase database table(s)
  - Store errors into separate database tables for logging purposes
- Video-to-text for television frames
  - Perform optical character recognition on text in television frames
  - Store text in searchable keyword/phrase database table(s)
  - Store errors into separate database tables for logging purposes

## Applicable Courses from Iowa State University Curriculum

- Com S 228 – Data Structures and Algorithms
- Com S 309 – Software Development Practices
- S E 339 – Software Architecture
- Com S 363 – Introduction to Database Management Systems

- S E 417 – Software Testing
- Com S 575 – Computational Perception

### New Skills/Knowledge acquired that was not taught in courses

In addition to the listed courses, this project required us to familiarize ourselves with topics such as audio manipulation, signal processing, optical character recognition, and more advanced examples of database management systems.

# Table of Contents

1	Introduction	5
1.1	Acknowledgement	5
1.2	Problem and Project Statement	5
1.3	Operational Environment	5
1.4	Requirements	6
1.5	Intended Users and Uses	6
1.6	Assumptions and Limitations	6
1.7	Expected End Product and Deliverables	7
2	Project Plan	7
2.1	Task Decomposition	7
2.2	Risks And Risk Management/Mitigation	8
2.3	Project Proposed Milestones, Metrics, and Evaluation Criteria	9
2.4	Project Timeline/Schedule	10
2.5	Project Tracking Procedures	10
2.6	Personnel Effort Requirements	11
2.7	Other Resource Requirements	11
2.8	Financial Requirements	11
3	Design	11
3.1	Previous Work And Literature	11
3.2	Design Thinking	12
3.3	Proposed Design	13
3.4	Technology Considerations	14
3.5	Design Analysis	15
3.6	Development Process	15
3.7	Design Plan	16
4	Testing	17
4.1	Unit Testing	17
4.2	Interface Testing	17
4.3	Acceptance Testing	18
4.4	Results	18
5	Implementation	18

6 Closing Material	20
6.1 Conclusion	20
6.2 References	20
6.3 Appendices	20

## List of figures/tables/symbols/definitions

### Figures:

Figure 1: Project Plan – The project plan for this specific project displayed in a Gantt chart. This is referenced in section 2.4.

Figure 2: Microservice Structure – This figure shows an image of the layout of the microservices that will be used in this project. This is referenced in section 3.7.

Figure 3: Microservice Calls – This image shows the microservice calls on Postman to convert the .wav file into text using the speech-to-text script. This is shown in section 6.3.

Figure 4: Script Output – The output after running the script which was called on the microservice application. This shows the time it took to translate the .wav file into text and how long the .wav file was. This is shown in section 6.3.

### Tables:

Table 1: Estimated Time – The table which displays the estimated time in hours needed to complete each task which is listed.wq This table was shown in section 2.6.

# 1 Introduction

## 1.1 ACKNOWLEDGEMENT

Our team would like to express our sincere gratitude to Bob Shapiro, Henry Bremers, and the team over at DigiClips for the continuous support of our project's research and implementation. Their constant encouragement, insightful comments, and hard questions have and will constantly help us create a better product that will be aid them in providing a more rounded experience for their client base.

We would also like to thank our faculty advisor, Ashfaq Khokhar, for the constant support through the project and for the insightful recommendations. His expertise in data intensive multimedia applications and knowledge of signal and audio processing has proven valuable on multiple occasions and will likely continue to do so throughout the course of our project.

## 1.2 PROBLEM AND PROJECT STATEMENT

DigiClips is a media content analysis company that records and extracts data from diverse types of media, such as television and radio, and stores this information in a searchable format. It aims to provide its clients a user interface that would facilitate searching of the database for keywords or phrases of user's interest uttered in audio or video clips. For example, a client may be interested in finding if their company name has been mentioned (along with its frequency) on television or radio within a given time frame.

**General problem statement** - The data currently being extracted from the television recordings is from the television network-provided closed captions only. This closed captions data often misses words or phrases spoken within the broadcast, causing a disconnect between the actual content of the broadcast and the searchable content provided. In addition to missed audio, closed caption data does not provide any means of searching the content of broadcasted frames themselves, where there is often visible text that denotes the current segment of news, breaking stories, etc. This information is, at the moment, lost within hours of recordings and extremely difficult to perform searches on.

**Proposed Solution** - This project will investigate existing solutions and develop efficient speech-to-text and video-to-text modules that will take television and radio recordings as its inputs and record the timestamp-location of keywords and phrases of interest in these recordings. The outputs of these modules will be organized in a database schema. The speech-to-text and video-to-text extraction capability will give the company an edge in the industry by granting them access to data that is currently not being tracked, providing more opportunities for clients to find any and all mentions of their keywords. The focus will be to develop near-real-time solutions that can scale with the number of audio and video recording streams. These modules will be integrated with other components in the system that include signal processing applications, databases, query and retrieval frameworks, and user-interfaces.

## 1.3 OPERATIONAL ENVIRONMENT

The operational environment of our project is simply a computer based in an office environment. The specific computer that our application will be running on is a custom-built system running Ubuntu 18.04. In terms of processing power, this computer boasts a very powerful AMD Ryzen 3900x CPU alongside an NVIDIA GeForce 210 GPU for minor graphics-based computations. The computer running our code will not be exposed to extreme temperatures or conditions thus the project will not be taking into account preventative measures for these anomalies.

## 1.4 REQUIREMENTS

- Functional Requirements
  - Speech-to-text system
    - Must be able to convert audio streams of spoken words into plain text.
    - Must accept mono and stereo audio recordings as input, processing all streams into their own results feed.
  - Video-to-text
    - System must detect multiple fonts/styles of text in video frames.
    - System must process text located within the bottom half of the recording's frames.
  - Output formatting
    - All system results must implement directly with existing DigiClips database schemas.
    - All system results must be processed to check for correct grammar and spelling.
    - Results should be indexed via their corresponding timestamp in the video/recording.
    - System errors should be traceable and identifiable for maintainability.
    - System errors must be stored in the DigiClips Errors database for querying and alerting.
- Non-Functional Requirements
  - System shall be built without utilizing any costly APIs/cloud resources.
  - System shall be built with documentation to explain usage and integration.
  - System should scale with the assumed amount of data present.
  - System should reliably output results within a reasonable timeframe.

## 1.5 INTENDED USERS AND USES

The intended users of this project are Bob Shapiro, Henry Bremers, and the DigiClips Media Search Engine. Bob Shapiro is the chairman of DigiClips Media Incorporated. Henry Bremers is the Senior Software Engineer in charge of managing DigiClips software. The DigiClips Media Search Engine is a front-end application that will be operated by DigiClips customers and clients and will be utilizing data produced by the project system.

## 1.6 ASSUMPTIONS AND LIMITATIONS

Assumptions:

- The program will only be processing up to 10 broadcast television channels.
- The program will be operating on new recordings as they are recorded rather than previously recorded broadcasts.
- The inputted television recordings will be of high resolution and high enough audio quality for accurate processing and output.
- The resulting program output will match the same database schema as the currently stored closed captions.

Limitations:

- Due to budget constraints, we are not able to utilize certain paid APIs for speech-to-text or optical character recognition.
- Our developed system must be computationally efficient and able to run on a relatively underpowered computer.
- The program must be able to operate quickly enough for customers to query data within 24 hours of recording.

## 1.7 EXPECTED END PRODUCT AND DELIVERABLES

The expected end product is a pipeline through which television and radio recordings will be processed to extract searchable data in the form of keywords and phrases spoken or visible through text in the video frames. As part of this end product, this pipeline must include a speech-to-text system and what we refer to as a video-to-text system which extracts words, letters, and numbers from the image frames of the recording.

The speech-to-text system must process audio identified as human speech into plain text phrases and keywords that, along with the timestamp of the spoken phrase, will be made searchable as part of a database. The system must be able to work with audio that comes from television and/or radio recordings performed by the current DigiClips recording backend.

Similarly, the video-to-text system will identify and process any words, standalone letters, and numbers that may be present within a given frame of the recording into searchable plain text words/phrases associated with the timestamp they were displayed in the recording to a searchable database to be used by the DigiClips front-end search engine service. The provided input to this system will be video recordings of television locally stored or frames of television passed through the pipeline as they are being captured/recorded.

These deliverables that make up the end product will be developed, thoroughly tested, and integrated into the existing DigiClips television recording backend by the project end date in December 2021.

## 2 Project Plan

### 2.1 TASK DECOMPOSITION

- **Build speech-to-text system.**
  - Planning
    - Determine what speech-to-text functionality is already present within the DigiClips codebase.
    - Research possible techniques for implementing speech-to-text using existing software.
  - Design
    - Outline possible speech-to-text system structure using planning and research.



- Collectively decide which outline is the most effective at meeting the problem requirements.
    - Development
      - Using the selected design develop the speech-to-text system as outlined in the Design phase.
    - Testing
      - Test the implementation to ensure the given requirements have been met.
      - Have our intended users test and report feedback on the project implementation.
  - **Build video-to-text system.**
    - Planning
      - Decide what video-to-text functionality currently exists in DigiClips software.
      - Research video-to-text processing to find the ideal solution to the problem.
    - Design
      - Build rough outlines of possible system structures.
      - Decide on which structure solves the problem most effectively.
    - Development
      - Using the chosen design build the implementation according to the proposed structure.
    - Testing
      - Rigorous testing of the implementation to ensure requirements have been met.
      - Receive feedback from our intended users on the state of the implementation and make sure their requirements are met.
  - **Integrate speech-to-text and video-to-text.**
    - Take completed speech-to-text and video-to-text systems and integrate them together to ensure ease of use for the user.
  - **Connect output data to DigiClips database.**
    - Format outputted data using timestamp and plaintext results according to database schema and save into database table.

## 2.2 RISKS AND RISK MANAGEMENT/MITIGATION

- **Speech-to-text task**
  - Speech-to-text processing will not be accurate enough to provide substantial value to the DigiClips business.
    - Probability: 0.2
    - Risk Mitigation Plan:
      - To mitigate this risk, we will put extra care into developing our speech-to-text system. With the current implementation we have already seen promising evidence that DeepSpeech will provide an effective solution to extract text from audio files. A future concern is that we won't be able to tag the data to denote when a sentence or word was detected in the audio file. To mitigate this, we will utilize our advisor's knowledge about audio tagging.

- **Video-to-text task**
  - Video-to-text system will be too processor intensive to be a realistic solution to DigiClips' problem.
    - Probability: 0.5
    - Risk mitigation plan:
      - Extra time will be spent in the planning and design phases of this task to ensure that we consider the limitations of processing large amounts of video data. To help, we will also limit search space.
  - System misidentifies words making it too inaccurate to be useful.
    - Probability: 0.5
    - Risk mitigation plan:
      - To mitigate this risk, we will compare our system's output with a standardized output to detect the system's accuracy. This will give us a better look at how effective our video-to-text software is.
- **Integration task**
  - The primary risk is that the speech-to-text and video-to-text systems won't integrate easily when being developed separately.
    - Probability: 0.5
    - Risk mitigation plan:
      - During the design and development stages of the speech-to-text and video-to-text systems we will plan for the future integration of the two systems. This will ensure that integration goes smoothly because we have planned in anticipation of this task. If the speech-to-text and video-to-text systems are designed with the capability to be linked together this risk should not affect our ability to complete the project.
- **Connecting to DigiClips database**
  - A risk could be that our project won't interface with the DigiClips database to effectively search the provided data.
    - Probability: 0.4
    - Possible risk mitigation plan:
      - To stop this problem from happening we need to familiarize ourselves with the current DigiClips database structure. This will help us design a method of storing the new data that is like the existing data. Then the task of modifying the search engine to effectively search this new data will be more of a simple task.

## 2.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

### Milestones:

- Complete the speech-to-text system.
- Complete the video-to-text system.
- Integrate speech-to-text and video-to-text on one complete program.
- Integrate with DigiClips database.

### Evaluation criteria:

- Achieve 80% accuracy on speech recognition and 95% coverage with microservice unit testing.
- Achieve 70% accuracy on video text recognition and 95% coverage with microservice unit testing.
- Process the speech-to-text for a video file within 75% of the file's length.
- Process the video-to-text for a video file within the length file.

## 2.4 PROJECT TIMELINE/SCHEDULE

### Rough Project Schedule:

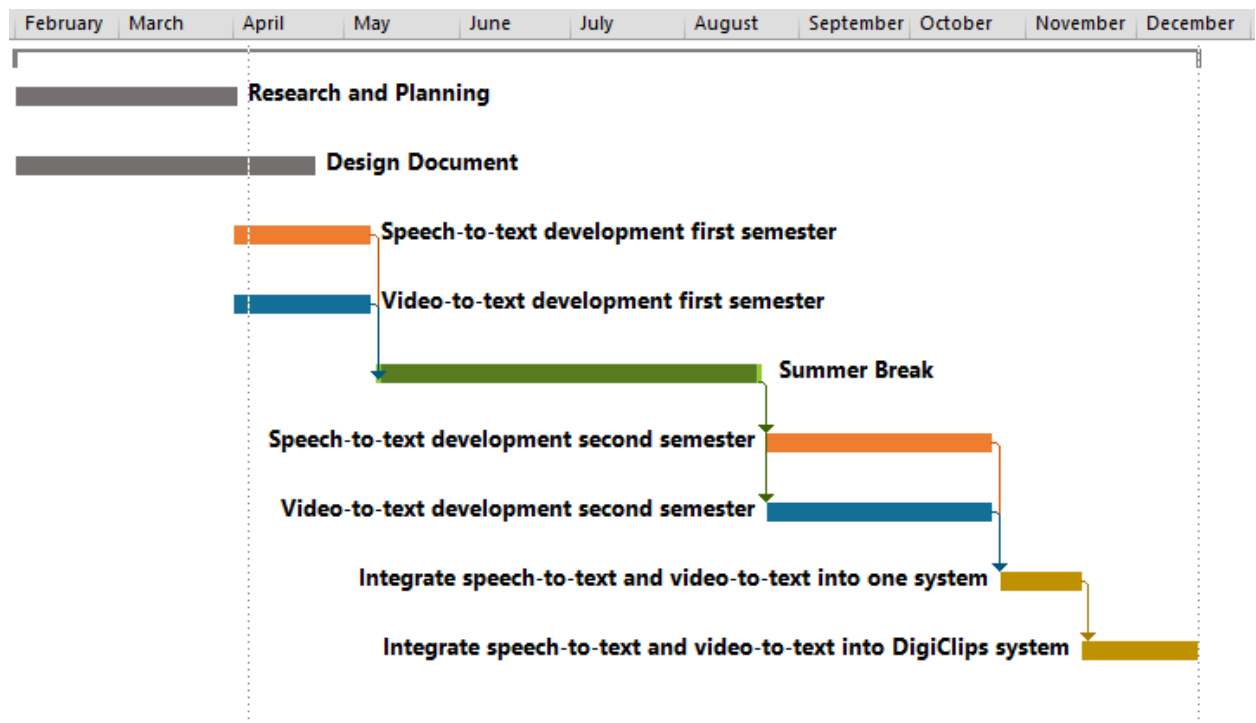


Figure 1: Project Plan

## 2.5 PROJECT TRACKING PROCEDURES

We plan to use multiple methods of tracking our project progress. Our main source of tracking progress will be GitHub as that is where the current codebase resides, and so when pushing to our repository we can see what was added, removed, or modified. We also plan to use communication channels like group messaging to ensure everyone is up to date with the current state of the project. Finally, we plan to have constant communication with our group as a whole and taking notes to have a better understanding of what should be expected.

## 2.6 PERSONNEL EFFORT REQUIREMENTS

Table 1: Estimated Time

Task	Estimated person-hours required.
Speech-to-text system	70 hours
Video-to-text system	100 hours
System integration	30 hours
DigiClips database integration	40 hours

## 2.7 OTHER RESOURCE REQUIREMENTS

We will not require any physical resources to complete this project. In terms of digital resources, we will need access to the existing code that DigiClips has pertaining to speech-to-text and video-to-text. We will also need access to the DigiClips database and backend code. This will ensure that we are able to integrate our solution with the rest of the DigiClips Media Search Engine.

## 2.8 FINANCIAL REQUIREMENTS

The only financial requirement we have for this project is that our application should not use paid libraries or APIs. Our project will make use of free, open-source software.

# 3 Design

## 3.1 PREVIOUS WORK AND LITERATURE

The idea of speech-to-text and video-to-text are not novel ideas, and many libraries and interfaces exist to perform these actions on multimedia recordings. Additionally, other groups have created similar applications for recording and pulling text specifically from television and radio stations.

Detailed in his Forbes article, Kalev Leetaru describes the process of using Google's speech-to-text API alongside natural language processing on television recordings to thematically analyze the segments of television. In this project, Leetaru mentions using the Google Cloud Speech-to-Text transcription to automatically generate a transcript for the video (Leetaru, 2019). In our research on the available APIs for speech-to-text we discovered that Google's implementation is considered as one of the top-of-the-line speech-to-text solutions currently available, especially when you consider its built-in grammar and spell checking alongside many other pre- and post-processing features that make the output into clear, accurate sentences and phrases. Unfortunately for our project, Google's speech-to-text solution is a paid one, leaving us to try and implement a lot of its functionality on our own. Our project aims to differentiate from Leetaru and Google's work primarily through the application of the resulting data. Rather than post-processing the data through natural language processing, we will be focused on formatting the resulting data to make indexed, query-able phrases stored in a database table.

Similar to Leetaru's implementation of speech-to-text for television, other groups have considered and possibly developed solutions for extracting speech data from multimedia sources. As seen in

their granted U.S. Patent, Daniel Barcy and Charles Statkus propose an implementation for extracting human-readable captioning for television and radio streams (United States of America Patent No. US6542200B1, 2001). In their design, an application runs using a live feed of television or radio audio and processes the audio input for gain control, audio filters, and finally into a speech-to-text converter that performs the data extraction. Their design also makes considerations for language translation processing, which they would perform on the direct output of the speech-to-text converter. A key difference between our proposed implementation and the one shown in the patent is the source of the audio input. In their design, the input is processed as a live feed, whereas in our implementation, we will be processing audio files after they have been recorded into a standard video format. This will integrate more smoothly with DigiClips existing television and radio recording systems and require less interference with their current applications.

Regarding video-to-text, one notable previous implementation comes from a team of researchers at University of Novi Sad, Serbia, discuss their implementation of using optical character recognition (OCR) on frames of television recordings. Their implemented system receives a frame as part of a live television feed, performs some image pre-processing including locating potential text, then processes those possible regions using an OCR solution. The output from the OCR solution is then used to verify the functionality of the television set as a means of testing hardware or software faults within the device. This implementation of OCR technology is very similar to our proposed design, since we will be attempting to locate text within each frame prior to OCR processing, which will greatly speed up our application and reduce the overall workload for our system. However, a few key differences in the implementation again stem from the input type and output formatting. In our application, we will be accessing frames of a television recording rather than grabbing frames from a live feed. Additionally, our application will be focused on making timestamp-indexed, query-able, phrases stored in a database table.

Lastly, some previous speech-to-text and video-to-text work has been made by other senior design teams for DigiClips. After some research and analysis of these previous implementations, we have found that the currently existing speech-to-text system is suffering from low accuracy while the currently existing video-to-text system suffers from extremely high processing times as well as low accuracy and consistency. Nevertheless, this existing work has the benefit of providing a point that we can work off rather than building from the ground up, as some parts of this project will already have a basic implementation. However, given our proposed architectural changes and our choice of processing libraries, for the most part we feel that this previously existing solution will not lend us much help, especially since it is not well documented.

### 3.2 DESIGN THINKING

We had performed decision making when considering and defining what data DigiClips is currently missing from their recorded channels. After discussing with Bob and Henry, who pointed out that a lot of data gets lost from text that is not necessarily spoken but appears on screen as a scrolling bar or other display, users would almost certainly want to be able to search for that text as it contains news updates, so we decided to add the video-to-text into our scope.

In addition to the speech-to-text and video-to-text, we had also generated the idea of lip-reading to text, where watching lip patterns to provide text as well. In the case of lip-reading to text we felt that this task was not very feasible due to technological limitations. Getting accurate text results from reading lips in a video is an immensely challenging task that might not be very accurate.

Rather than attempting to do something like this we have decided to focus on speech-to-text because it is a simpler task that fulfills the same goal. Having a good speech-to-text system would replace the need for a lip-reading system.

During the ideate phase of our design thinking process we came up with some design decisions that we ended up not going with due to limitations. Primarily, using paid APIs and services to achieve our goals. For example, Google has a high-quality speech recognition API that would work great for our use case. However, this is a service that requires payment. For the standard speech-to-text service it is \$0.006 / 15 seconds of audio. Running eight television channels through this constantly for a month would rack up charges close to \$8,600. This kind of pricing model is not feasible for DigiClips to even consider given how much television and radio they constantly record.

Another decision we made during our design thinking process was related to programming languages. Initially, we thought that C or C++ might be the best for this kind of application. However, upon further research, it became clear that Python seems to have the most up-to-date and well-maintained libraries when doing optical character recognition and speech recognition. Using C or C++ would give us some speed, but libraries to perform this kind of analysis aren't as available as Python.

### 3.3 PROPOSED DESIGN

The first method of solving this problem that we came up with was to use C or C++ to develop an app that performs Optical Character Recognition (OCR) and speech recognition on video files. We initially thought to use C or C++ because of the speed that those languages offer. Having extra speed and efficiency would give our application an edge since DigiClips has a massive amount of data to analyze. Eventually, we discovered that there are not many libraries for OCR and speech recognition written for C and C++. There are some options, but they are difficult to use, not well supported, and not user-friendly.

Once we realized that C or C++ might not be a viable choice, we investigated possibly using Java. We were a bit hesitant to try Java because it is well known to be much slower than other languages due to its built-in garbage collection and inefficient data access. However, there is a well-supported speech recognition library, CMUSphinx, designed to be used in Java. We found that using this library to perform speech recognition was effective at fulfilling the speech-to-text part of our project's requirements.

At this point, we had already found that Java was not a good choice for the video-to-text portion of our project, so we began to consider using a microservice-based architecture. Since speech-to-text implementation in Java worked well, we could run speech-to-text in Java and video-to-text in another language. This design would utilize one of the primary benefits of using microservices, simple communication between different programming languages.

Next, we investigated using Python since our team had already done some work using Python's popular OCR software, Tesseract. Python seemed like a viable choice for doing the video-to-text portion of our project. In our research, we saw a speech recognition library developed by Mozilla called DeepSpeech. This library seemed like a viable choice because it's open-source and well maintained, so we set it up and tried it out with some of our sample data. After some initial testing, we realized that DeepSpeech had a similar level of accuracy when compared to CMUSphinx and

was over twice as fast. This discovery made DeepSpeech the obvious choice to ensure our application is as efficient as possible.

At this point, we have decided on using Python both for our video-to-text system and for our speech-to-text system. These two systems must be as efficient as we can make them, so we must take care when designing their functionality and their interactions. We have also decided that a microservice architecture will be an effective solution to this problem because it is flexible and resilient to crashes that could hamper DigiClips' business model. Also, using existing REST API technology allows for requests to be made to each service in parallel. Parallel processing using REST API technology will enable DigiClips to keep up with incoming data without complex process management systems.

This design for our application will have three different services that will interact with one another: a central driver service, a speech-to-text service and a video-to-text service. First, the central driver service will process each request as they come in, sending the audio to the speech-to-text service and the video to the video-to-text service. Once the driver receives the results from both sub-services, the data will be ready to be entered into the database. Then, the speech-to-text service will process the audio of the video clip to detect speech. Lastly, the video-to-text service will process each frame of the video clip to find text.

This project design will capture all the necessary processing into one application to streamline extracting data from the video files that DigiClips collects. Then, we will store the data in a searchable manner on the DigiClips database to vastly increase the amount of data that the search engine's users can access.

### 3.4 TECHNOLOGY CONSIDERATIONS

The current limitations are mainly on the computers used to run the programs that will be written. Video-to-text is a taxing process, and the CPU can only handle so much, so checking the entire video frame every frame would severely limit performance. In addition, the CPU must also handle the recording of the normal speech-to-text which also is rather intensive, though not as much as video-to-text. Performing both would be too much to handle, and so limiting frames and frame sizes will be needed.

Possible alternatives would be to attempt code optimization and reduce the code as much as possible while still performing the same job. Removing unnecessary and rewriting inefficient code may help lessen the burden these processes carry. As well, it may show that the current computer strength is limited, and in need of an upgrade. A newer computer, or adding another device, would help to split the load. Otherwise, we will need to decide how to limit our code to run smoothly.

We know that the computer that our application will likely run on has a Ryzen 9 5900X 12-core CPU. This is a very high-end CPU and will give us a substantial amount of processing power to work with. That computer also has a Nvidia GeForce 210 GPU. This is unfortunate because this GPU is low-end and does not offer much processing power. GPU processing is something that can make a significant difference in the kind of calculations we will be doing in both speech-to-text and video-to-text. Thus, we must take the lack of an effective GPU into consideration when designing our application.

### 3.5 DESIGN ANALYSIS

So far, our proposed design has been moderately successful at fulfilling our speech-to-text and video-to-text goals. The proposed microservice architecture has not been completely implemented at this point, but the video-to-text and speech-to-text systems are functioning on their own to some extent. Now, we are focused around finding the best way to perform the speech-to-text and video-to-text.

In terms of the speech-to-text system, we have discussed splitting each audio file into chunks and processing each chunk separately as an option to increase the efficiency and accuracy. Also, we are working with several different grammar and punctuation libraries to ensure our speech-to-text output is as readable as possible.

More research is currently being performed on our proposed design to evaluate its effectiveness in a few key areas. We believe that the proposed iteration of our design is effective for several reasons. Firstly, the design is functional and capable of producing the results we wish to acquire, as seen in our first round of design testing where we implemented a barebones system and compared the output to our expected result. Secondly, the design is maintainable. Almost all the source code that our client is familiar with and typically uses is written in C or C++, therefore the lead developer in charge of maintenance will be able to edit and maintain our program as it progresses. Our design also makes use of API calls which will be easy to swap in and out should a better API solution be developed. Finally, the components which we are still evaluating design effectiveness on are reliability and feasibility. In terms of reliability, we are confident that our system will be able to handle the expected amount of traffic, however we have been thus far unable to generate the amount of traffic our system will be expected to maintain as we do not have full access to the television and radio recording devices at DigiClips. In terms of feasibility, our current concern with the design is primarily focusing on the selection of a speech-to-text and optical character recognition API. We have noticed some difficulty in obtaining an accurate model for both tasks that is open-source and free in cost.

### 3.6 DEVELOPMENT PROCESS

We plan to use Agile programming to work on our project. To further this, we will most likely be going on Feature Driven Development (FDD). We want to work in small pieces, have those pieces working as intended, then work on using those pieces to create larger parts and connect them all together. This will make sure that the basic components work, and that connecting the components will be easier. Such as, working to make sure we can receive audio from a file, then make sure we can extract speech text, then checking that it can work on recorded examples with accuracy and finally working on live audio.



### 3.7 DESIGN PLAN

We will have a couple of different services that interact with each other to perform the necessary functions. Here is the proposed microservice structure:

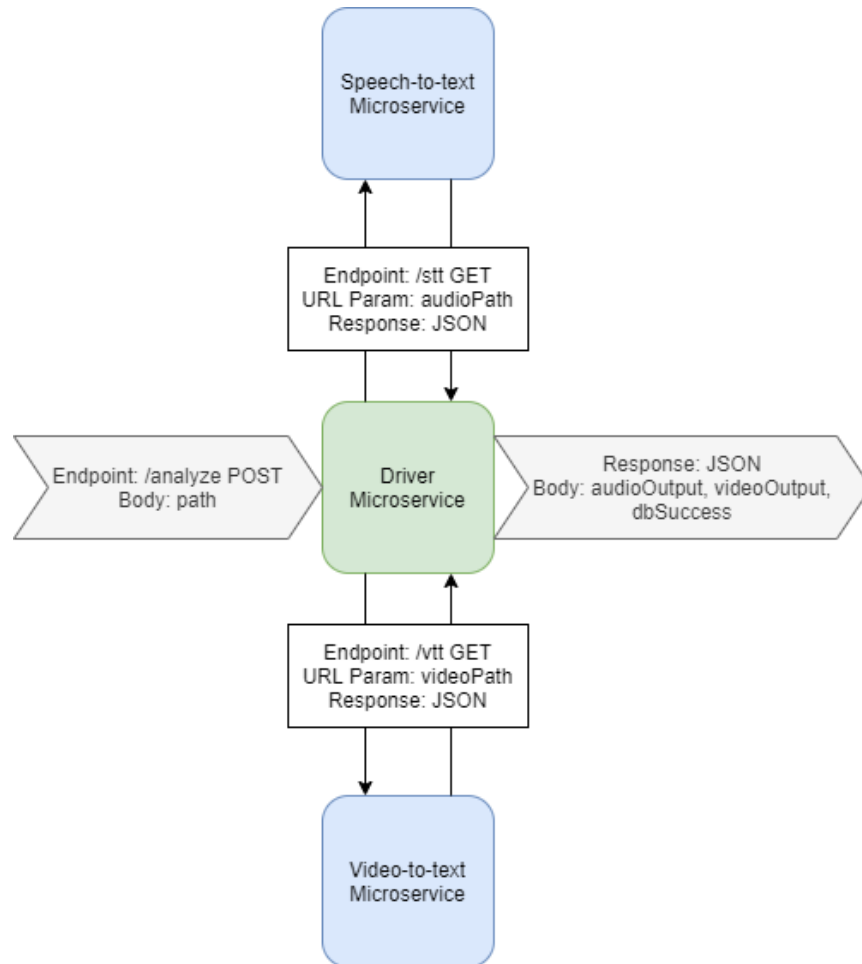


Figure 2: Microservice Structure

#### Driver Microservice

The entry point to this pipeline is through the Driver Microservice. **POST** Requests will be made to the `/analyze` endpoint of this service. A file path is needed in the body of a request to supply a video file to analyze. The Driver Microservice will process the file at the given path to prepare it for processing by the Speech-to-text Microservice and the Video-to-text Microservice. The Microservice will strip the audio from the video and save it as a `.wav` file. Then the paths to the audio and video will be passed to the Speech-to-text Microservice and Video-to-text Microservice, respectively.

#### Speech-to-text Microservice

Requests made to the Speech-to-text Microservice will be **GET** requests with a URL parameter containing the path to the audio file. The microservice will process this file to extract text from the audio. The extracted text will be formatted **JSON** containing tags that identify the timestamp for

each piece of text. This JSON will be sent back to the Driver Microservice to be added to the database.

### **Video-to-text Microservice**

Requests made to the Video-to-text Microservice will be GET requests with a URL parameter containing the path to the video file that needs to be analyzed. The microservice will analyze each frame of the given video, looking for text on the screen. This text will be extracted from the video and saved to be stored in the DigiClips database later. The microservice will return the extracted text within formatted JSON containing tags that identify the timestamp for each piece of text in the video. Once this text is returned to the Driver Microservice, the app will store it in the database for searching.

## **4 Testing**

### **4.1 UNIT TESTING**

For unit testing, we will be utilizing the built-in test suite for Python (unittest) as well as using the popular pytest suite. These test suites, along with the necessary program models, will be the main parts of the unit tests. Unfortunately, due to the nature of our application and the variation between input (video/audio recordings) and output (plain text), it is difficult to perform unit testing on the speech-to-text and video-to-text component of our software. Apart from transcribing hours of audio and video, there is little test validation that we can do for our primary systems. However, there are a few key areas where we will be primarily implementing unit testing to ensure an accurate result. We will create a series of tests that confirm the smaller and more testable individual functions, such as grammar and punctuation. Alongside this, we will be testing the speech-to-text portion of our application through manually creating a small set of test cases for a small variety of video files, which we will manually transcribe and compare speech-to-text output for.

Given that we are developing our project in an Agile sprint-style development cycle, we will be performing regression testing as part of our unit testing to ensure that as features are added, previous implementations are not affected. To do this, we will be keeping a large set of test cases for the system as a whole and continuing to run these preliminary test cases as new features are added. Larger tests that incorporate multiple units or functions of the program will be used in higher level tests, such as interface tests.

### **4.2 INTERFACE TESTING**

Our interface testing will primarily consist of confirming the microservices are working as expected across various input. Since we will be creating small microservices using Flask servers, interface testing will be slightly different than standard. Testing an endpoint-based microservice is not as straightforward as many class or object-based testing, primarily because the microservice cannot be simply instantiated within the testing file. Therefore, to test our various microservices, we will be performing API testing using the endpoints. This requires performing HTTP requests with various input parameters and checking for accurate responses and error reporting. We can then check

additional features such as multiple simultaneous calls and checking various lengths of audio to ensure no information is lost.

A large part of interface testing will be performed once we begin development on integrating the speech-to-text and video-to-text into one system. At this point we will not only be checking for accurate and functional microservices but will also have to test for correct database table usage. Given that our system will be writing output data to multiple distinct database tables with perhaps unique schema for each table, we will need to perform testing to ensure that the data format is correct for columns of the table as well as checking to make sure that the content for each column is accurate based on the raw output from the speech-to-text and video-to-text microservices. These forms of testing are referred to as schema testing and column testing, respectively.

### 4.3 ACCEPTANCE TESTING

Our acceptance testing will be done by providing the DigiClips group with our results of both the accuracy and speed of our program. The most important part of this will be the accuracy of the code. Inaccurate output will mean that users will not be able to correctly search for snippets of text and will therefore not be of use. While making sure text is accurate, we must also ensure that the output is done in a timely manner. New input will be provided every 34 minutes, and so these “chunks” of data must be completed before a pileup of input occurs. Due to limiting constraints regarding processing on the client system, we will need to make sure that acceptance testing regarding processing time takes place on the client’s system and not our own. We will involve our clients by providing them this data and discussing acceptable accuracies and times for processing. In addition to this hard data and functional requirements, we will also make sure non-functional requirements are also met to their satisfaction.

### 4.4 RESULTS

As of now, our testing results are minimal. Given that we are just now finalizing the design portion and beginning work on the development portion of our application, testing will very soon become a priority as we validate features and sprints along our development cycle. As of this moment our only testing has been informal testing to ensure that our basic data pipelines are working to utilize different APIs and libraries for speech-to-text and video-to-text.

Earlier preliminary testing was performed to determine which programming language would be best suited, which we prioritized early on due to it being a constraining factor on available APIs and libraries. We are also in the process of testing multiple speech-to-text APIs that have thus far had a fairly large range of output that we must formally analyze through string comparison. Throughout the entire process, we have been considering the overall functionality of the program, to ensure that the requirements are met and that we receive the expected results.

## 5 Implementation

Our implementation plan is focused on using Python to build the microservices that will make up our app. Python will enable us to use cutting-edge OCR tools and speech recognition models. While it may not be as fast as other languages like C or C++, we think Python will be a great solution to fulfill our requirements.

To build our REST API microservices, we will use the popular Python framework Flask. Flask offers a straightforward interface for building both simple and complex web APIs. There is minimal configuration and boilerplate needed to get a Flask web service up and running, which will enable us to keep our services slim and straightforward. The REST API structure enables our services to process requests simultaneously. Parallel processing helps our services process even more data within a given time. With Flask, the programmer doesn't have to create complex process management systems to facilitate parallel processing. The framework will handle everything automatically.

Our speech-to-text system uses the DeepSpeech library, a library developed by Mozilla using Tensorflow, a Python machine-learning library. Mozilla has gone to extreme lengths collecting thousands of hours of crowd-sourced voice recordings with their project Common Voice. They then use all this data to train their DeepSpeech recognition model that we use in our application. Using these existing libraries and systems enables our team to create an efficient and accurate application without collecting mountains of data and building machine learning models. Thus, we can make much more progress than we would without utilizing open-source libraries.

An important consideration for our speech-to-text service is how we send data into the DeepSpeech model. Sending in an entire 30+ minute audio file can strain the computer the service runs on and takes a long time. Through some testing, we have found that breaking the audio file into chunks and passing each piece into the model on its own is a much more effective strategy. However, we must consider how to split the audio file. If we divide in even sections, for example, into fourths, there is a chance that we will cut an essential word in half and lose that data. A way around this could be first to detect silent portions of the audio and then split the file by those sections. This method will mitigate the possibility of breaking the file in the middle of a word. Currently, this is the implementation strategy we are using for our speech-to-text system. As we do more testing and research, our approach may change slightly, but this is our starting point.

On the video-to-text side, our current implementation exists primarily as a prototype data pipeline through which we can continue to test different OCR solutions to find one with acceptable accuracy. Currently, the system reads in a video file using OpenCV's VideoCapture object and begins to step through the video frame-by-frame. Some preprocessing is performed on each frame such as binary thresholding and canny edge detection, which serve to highlight the contrast between text and background as well as determine more noticeable edges for the text. After pre-processing, the frame is passed into Google's TesseractOCR, a free and open-source optical character recognition library, which extracts text from the image and parses it into a string. Once the output is received, some minor post-processing is performed to trim whitespaces, eliminate excessive punctuation, and more. As each frame is processed, a counter is kept keeping track of the frame's location since the beginning of the video which, along with the framerate of the video, is used to determine the frame's timestamp to be paired with the text output, currently as a basic tuple. Therefore, currently the system is meeting the requirements of processing text from a given video and indexing the result with a timestamp, however the system does not act as a microservice, and hence will need to be encapsulated into a Flask endpoint to match the proposed system architecture. Additionally, work will have to be done to determine the best pre-processing and post-processing steps to achieve the highest accuracy across various fonts, typefaces, and colors.

## 6 Closing Material

### 6.1 CONCLUSION

Our team has worked closely with DigiClips to create and develop an initial plan to complete the project we have been assigned. Our goal is to create an element of speech-to-text and video-to-text that will use self-developed and open-source software, which will in the end be incorporated into the DigiClips overall system, allowing them to translate any television recordings into something that is searchable using their own search engine. Currently, after doing research and different testing on each element, the best plan of action that we currently have is to develop each element individually, test the element that was created, then incorporate them into one product which can be used by the system in place at DigiClips. This solution, using python with different open-source software, will be the most reliable option with the given requirements.

### 6.2 REFERENCES

- Barcy, D. J., & Statkus, W. C. (2001, August 14). *United States of America Patent No. US6542200B1*.
- Kastelan, I., Kukolj, S., Pekovic, V., Marinkovic, V., & Marceta, Z. (2012, September 22). Extraction of text on TV screen using optical character recognition. *2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics*.
- Leetaru, K. (2019, June 8). *Using Google's Speech Recognition And Natural Language APIs To Thematically Analyze Television*. Retrieved from Forbes:  
<https://www.forbes.com/sites/kalevleetaru/2019/06/08/using-googles-speech-recognition-and-natural-language-apis-to-thematically-analyze-television/?sh=460cbd87783b>
- randall. (2017, December 17). *FM2TXT: Automatically Perform Speech To Text on FM Signals*. Retrieved from RTL-SDR: <https://www.rtl-sdr.com/fm2txt-automatically-perform-speech-to-text-on-fm-signals/>

### 6.3 APPENDICES

#### Appendix A: Helpful links

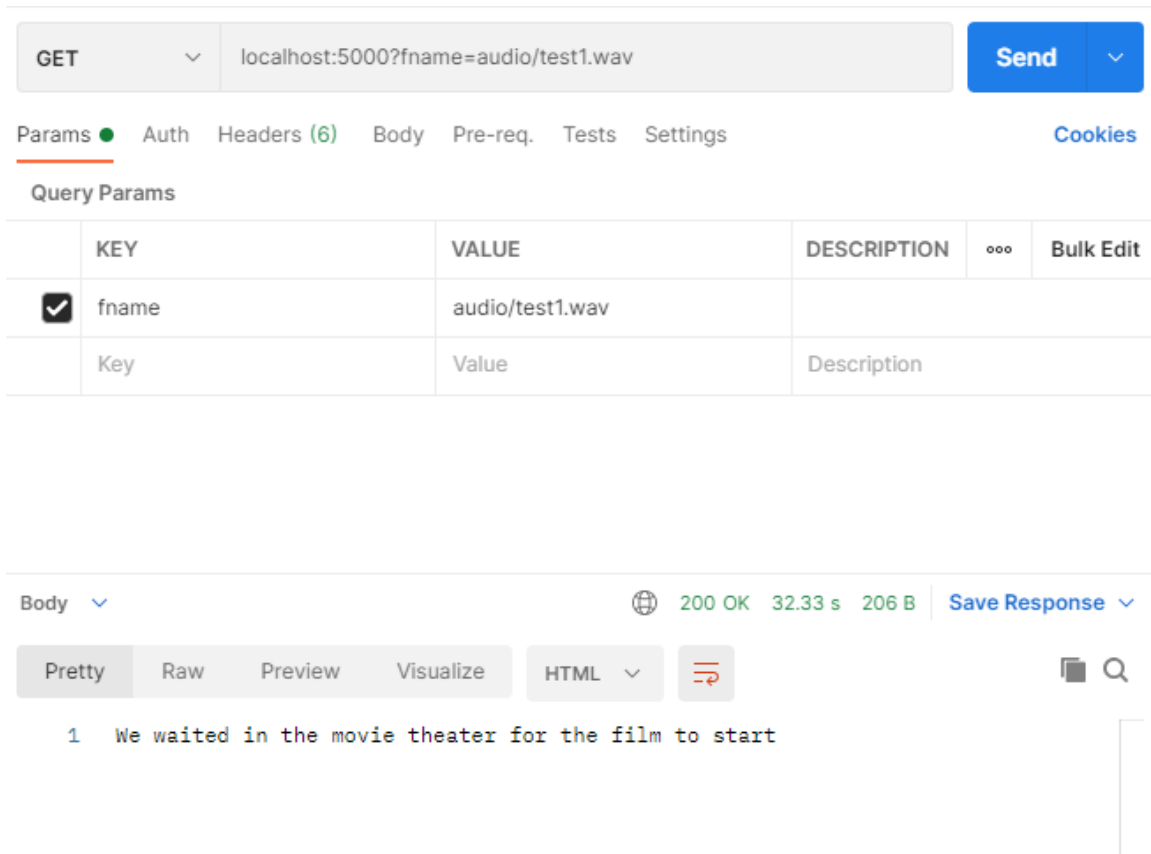
Here are some links that provide more reading on the technology that we are utilizing in our project:

- I. Mozilla's DeepSpeech: <https://deepspeech.readthedocs.io/en/ro.9/?badge=latest>
- II. Mozilla's Common Voice Project: <https://commonvoice.mozilla.org/en>
- III. Research paper that inspired DeepSpeech: <https://arxiv.org/abs/1412.5567>
- IV. Flask framework: <https://palletsprojects.com/p/flask/>
- V. Flask web service testing: <https://flask.palletsprojects.com/en/1.1.x/testing/>
- VI. Google's Tesseract OCR: <https://opensource.google/projects/tesseract>

#### Appendix B: Useful images

The following are images to show current output as well as other useful information:

- I. The output using speech-to-text with an input of a .wav file which says, “We waited in the movie theater for the film to start.” The call and output are all done using Postman:



The screenshot shows a Postman interface for a GET request. The URL is localhost:5000?fname=audio/test1.wav. The query parameters table is as follows:

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	fname	audio/test1.wav			
	Key	Value	Description		

The response body is shown in the 'Body' tab, displaying the text: 1 We waited in the movie theater for the film to start

Image 3: Microservice Call

- II. For the output show in Appendix B: Section I, below is an image taken from the scripting software with the output amount of time taken to transcribe the .wav file:

```
Transcribed text for 3.66s long file in 4.28s.
```

Image 4: Script Output