Chapter Summarization

Virginia Tech, Blacksburg, VA 24061

Instructor: Prof. Edward A. Fox

Client: Bipasha Banarjee

Aidan Simms, Manasi Peta, Joe Grilli, Nandha Chokkan
# Table of Contents

1.0 Table of Figures ......................................................................................................................... 3
1.1 List of Tables .............................................................................................................................. 5
2.0 Abstract ...................................................................................................................................... 6
3.0 Introduction ................................................................................................................................. 8
  3.1 Problem .................................................................................................................................... 8
  3.2 Motivation ............................................................................................................................... 8
  3.3 General Approach ................................................................................................................... 9
4.0 Requirements .............................................................................................................................. 13
5.0 Design ......................................................................................................................................... 15
  5.1 High Level Backend Design ................................................................................................. 15
  5.2 Frontend Design .................................................................................................................... 16
6.0 Implementation .......................................................................................................................... 24
  6.1 Backend Implementation ....................................................................................................... 24
  6.2 Frontend Implementation ....................................................................................................... 26
7.0 Evaluation .................................................................................................................................... 31
8.0 User Manual ............................................................................................................................... 33
  8.1 Use cases / Tasks Supported ................................................................................................. 33
9.0 Developer Manual ...................................................................................................................... 36
  9.1 Figures .................................................................................................................................... 36
  9.2 Inventory ............................................................................................................................... 39
10.0 Lessons Learned ....................................................................................................................... 40
  10.1 Timeline/Schedule ............................................................................................................... 40
  10.2 Problems ............................................................................................................................. 41
  10.3 Solutions ............................................................................................................................. 44
  10.4 Future work .......................................................................................................................... 47
11.0 Acknowledgements ................................................................................................................... 49
12.0 References ................................................................................................................................. 50
13.0 Appendices ................................................................................................................................. 54
  13.1 Appendix A: Methodology .................................................................................................... 54
  13.2 Appendix B: First Batch of ETDs ......................................................................................... 59
  13.3 Appendix C: Second Batch of ETDs ...................................................................................... 59
1.0 Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Object Detection Performed on a PDF</td>
</tr>
<tr>
<td>Figure 2</td>
<td>List of Elements Excluded in Document for Data Extraction</td>
</tr>
<tr>
<td>Figure 3</td>
<td>High Level Pipeline Design</td>
</tr>
<tr>
<td>Figure 4</td>
<td>The Original Experimenter Page Design</td>
</tr>
<tr>
<td>Figure 5</td>
<td>The Proposed Experimenter Page Design</td>
</tr>
<tr>
<td>Figure 6</td>
<td>The Proposed Experimenter Page After a File Within Page Limit is Uploaded</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Experimenter Page After a File Outside Page Limit is Uploaded</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Curator Page Before a File Upload</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Curator Page After a File Upload</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Summary Database Page</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Displaying Results on Summary Database Page</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Minor Frontend Changes</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Original Sidebar and Proposed Sidebar</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Backend Pipeline Libraries for Data Extraction</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Curator Page Alert For Successful Upload</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Curator Page Alert for Exceeding Page Limit</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Experimenter Page Alert for Exceeding Page Limit</td>
</tr>
<tr>
<td>Figure 18</td>
<td>The Redesigned Experimenter Page</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Use Cases for Readers</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Use Case for Authors</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Development Branch in GitLab</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Generated Summaries Directory Structure</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Summarization Model Choice</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Chunk Size Variable</td>
</tr>
</tbody>
</table>
Figure 25: Summarization Chunking Logic

Figure 26: Rough Overview of Timeline and Current Progress

Figure 27: Timeline Containing Technical Milestones

Figure 28: Integration Error When Setting Up Database Connection

Figure 29: User: Readers Breakdown of Goal and Subtasks

Figure 30: User: Authors/Researchers Breakdown of Goal and Subtasks

Figure 31: Workflow Diagrams of Services Provided
1.1 List of Tables

Table 1: Sample Data from the “Summary” Table of the Database ...........................................29
Table 2: Implementation Specific Information for Every Service .............................................57
2.0 Abstract

A thesis is the amalgamation of research that serves as the final product of a graduate student’s knowledge about the information they learned throughout their graduate research. A dissertation is a graduate student’s opportunity to present their original research that they have worked on during a doctoral program to contribute new theories, practices, or knowledge to their field. Theses and dissertations represent the culmination of research of students and therefore can be extremely long. Electronic theses and dissertations (ETDs) are the digital versions of theses and dissertations so that the research and knowledge explored can be more accessible to the world.

ETDs typically contain an abstract describing the work done in the document. However, these abstracts often don’t help readers identify chapters of interest. There is no happy medium between getting essentially no information from generic abstracts and reading through a dense paper. This is an issue on a global scale.

We created chapter summaries of ETDs which aim to help readers decompose and understand the documents faster. We make use of existing machine learning summarization models, specifically Python packages and language models, to help with the summarization. Part of this project is to create a dataset we can work with to create and test our summarization model on. This summarization dataset has been created by annotating the chapters from 100 ETDs (after chapter segmentation). We want to be as diverse as possible, while also being able to pick up on patterns, which is why our ETDs are from a plethora of fields.

We have implemented a data extraction pipeline that builds on work done by the Object Retrieval Code from Aman Ahuja et al [1]. Based on this we have created a summarization framework that accepts the chapter text as input and generates chapter summaries that are integrated into the given base front-end website application.

We have completed 4 summarization scripts that utilize pretrained models from Hugging Face which take in the data extracted from the chapter, and output a summary of the input data. The four models we used were BART [22], BigBirdPegasus [23], T5 [10], and Longformer Encoder Decoder (LED) [14]. We were able to use these scripts on all the chapters that we manually segmented to get summaries of all the chapters. We organized these summaries, based
on what model we used to obtain them, in our GitLab repository. We used these summaries to populate a database which was intended to be used for the search functionality of our frontend application. There is more about the specifics of the backend and frontend in Section 6.

We gained a holistic understanding of working on a full-stack project. On the backend portion, we learned how to use existing libraries and resources like Pandas, PDFPlumber [17], and WordNinja [19] to extract and format data from an input source. We also learned how to use resources like Hugging Face to understand natural language processing models and the pros and cons of various types of models. By creating scripts to utilize such models for text summarization, we were able to learn the nuances of working with pretrained models and understand how that can affect our product. For example, if a model was pretrained on a massive text repository, then it had better chances of recognizing more uncommon words in ETDs. On the frontend portion, we gained experience using React and JavaScript to create a functioning website. We also learned the process of understanding, dissecting, and updating a codebase we inherited from another team. We learned how to create and populate a database in PostgreSQL (commonly referred to as Postgres).
3.0 Introduction

3.1 Problem

ETDs are typically very dense documents that are prefaced with an abstract. However, these abstracts are general, covering the entire document, which means they do not give meaningful information or summaries at the chapter level. There is nothing that makes it easier for readers to find parts of the documents they might be interested in.

There are a multitude of issues that come from this lack of chapter summaries. Since ETD abstracts are for the entire document, it is very difficult to pinpoint the depth of the research and how many fields are involved in the creation of the knowledge of the ETD. With the current systems, there is only general document level metadata. This makes searching and categorization very difficult for parts of documents. This is a problem for people who need this information when looking for ETDs. It is especially difficult when looking for interdisciplinary works.

3.2 Motivation

The motivation for this project is to make ETDs more accessible and easier to search, for scholars as well as the general public. In academia, ETDs are extremely important as they allow students to communicate their knowledge on a particular subject matter, not only to their professors, but to future students across the world. However, ETDs can often be long (hundreds of pages), making it hard for readers/stakeholders to identify portions of interest. Our project aims to provide an easy-to-use interface for curators to input chapter text and receive accurate summaries that later can be found by other users. This can be extremely valuable for users like students who are new to a certain topic, or for researchers wanting concise, accurate information about a certain subject area.

The benefits of this are that stakeholders can more easily and quickly review multiple sources of information, making their information gathering more efficient. In addition, this can allow for those who are new to a subject area to extract knowledge from works that contain unfamiliar technical jargon. All in all, this project has the potential to allow dense reading...
material to be digested easily, as well as facilitate faster, more efficient sharing of information between stakeholders like students and the general public. By accomplishing these goals, ETDs can be made accessible to a wider range of individuals than previously possible.

3.3 General Approach

For our project, our first task was to manually segment around 50 ETDs from different STEM disciplines by chapter. These ETDs were chosen by our client, who upon completion of this initial task, tasked us with collecting another batch of ETDs from different, non-STEM disciplines from a database that she provided us. The first of two obstacles associated with this was that for our processing to work, the ETDs generally needed to be from after roughly 2010, because many earlier ETDs were not “born digital”. Some ETDs are scanned from paper documents, and so the PDF is a set of page images; “born digital” documents were created electronically, with formats like PDF resulting from a conversion from Word or LaTeX or another electronic version. Newer ETDs generally are born digital, while those earlier (e.g., before 1997) are generally created by scanning. There are exceptions where some ETDs are still scanned-in documents, as opposed to digital documents. The second obstacle was that some of the documents described in the database were not accessible by our client, since the database contained metadata from several thousand ETDs from outside universities, and the full version of a document is not always readily available. After 10 suitable ETDs were selected from each of the disciplines of Psychology, Architecture, English, History, and Music, they were segmented in the same way as the previous collection of ETDs. Once the initial task of collecting our test data for our system was complete, our client instructed us to review and study the Python libraries that will be utilized by our backend pipeline, along with other possible alternatives, to determine if they should be used in place of the existing skeleton for the pipeline.

The next task we were given was to continue to research and experiment with the pipeline using pages from ETDs of the different disciplines. This was done to see any potential roadblocks with the text extraction process. More specifically, this project requires efficient isolation of relevant text from an ETD, so elements such as chapter names, headers, footers,
equations, table data, and citations need to be excluded before summarization can occur. The skeleton pipeline was used to categorize page elements and determine what text is relevant to the task of summarization. After a week of experimenting, our team found that the current pipeline has data extraction issues due to unique table formats within non-STEM ETDs.

Recently, a two-stage anaerobic membrane bioreactor (AnMBR) consisting of thermophilic pretreatment (55 ± 2 °C; 20 – day HRT) as Stage 1 and a mesophilic AnMBR (37 ± 1 °C; 45 – day HRT) as Stage 2 demonstrated promise for treating UV disinfection-interfering substances in landfill leachate. To gain a better understanding of the mechanistic role of each treatment step, an identical single-stage mesophilic AnMBR (37 ± 1 °C; 45 – day HRT) was compared over a 6-month period, without the preceding thermophilic step. It was found that, on average, the two-stage system removed 55% of the organic carbon present in raw landfill leachate, while the single-stage AnMBR only removed 15%. Further, the two-stage system was able to remove nearly 50% of the UV254 absorbance, while only 16% was removed by the single-stage AnMBR. These results support the hypothesis that mimicking the conditions of long-term landfilling in an engineered system can help improve degradation of biorefractory organic carbon compounds, particularly humic and fulvic acids responsible for UV-quenching and reduction in UV254 absorbance and effluent disinfection performance over time. Shotgun metagenomic DNA sequencing and qPCR analysis of microbial communities in samples of raw leachate and from

Figure 1: Object Detection Performed on a PDF adapted from [1]
Our group utilized a PDF Object Detection tool that was developed by Aman Ahuja et al. [1] and can be used to help with our project. This tool outputs bounding box coordinates of the typical attributes in academic PDF files in addition to a visual representation of these elements as seen for one page in Figure 1. Each element is identified with a different colored box and a corresponding element title label, which is in the same color as the drawn box. We can look at the bounding box information given separately and have a visual to which we can match this information.

The intention was to obtain the bounding box information of unhelpful information such as page numbers, equations, and images/figures, which would be excluded in the text extraction process. Further inspection found that there are major obstacles with the dimensions of the bounding boxes that are causing the extraction of text to be incomplete or to not fully exclude table data. PDFs were rendered with the pdf2image [11] library in the Object Detection tool with a different coordinate system from the PDFPlumber [17] library, so bounding boxes generated with pdf2image [11] were invalid. After taking a deeper look into the issues, we found a solution in changing the scale of the bounding boxes. We could then tackle the creation of the summarization pipeline.

The summarization models are created using HuggingFace’s open-source natural language processing models. We used these state-of-the-art machine learning models to develop multiple summarization model scripts.

In parallel with the efforts with data extraction and summarization aspects of the NLP pipeline, our team had been building our web interface by pulling elements from a previous web interface provided by our client. This skeleton was modified and built upon further to relate to our project. Unfortunately, the deliverable went uncompleted, which is discussed more later. Here, we will discuss what we attempted to accomplish with our implementation.

Our goal was to create a web interface that would allow for users to upload a chapter of an ETD, and then choose a summarization model to run on the chapter. In theory, after our system ran the summarization model on the extracted text from the chapter, a suitable summary would have been output for the user to view. This process would have also allowed users to easily extract information from the summary they requested. This summary then would be added
to a database of summaries that have been previously processed by the system. The database of generated summaries would give the user the flexibility to return to the summary they requested at a later time without needing to wait for the summary to be generated, and give them the ability to review other summaries that were created by our system. The summary database would have been searchable through keyword matching to an ETD’s title, abstract, or its generated summary. Results could also be filtered by things such as the academic discipline the ETD falls within, or the summarization model used.

With the implementation of the data extraction and summarization pipeline and a functioning frontend, we would’ve connected the parts together with a master script to make sure users can use the summarization models to create new downloadable summaries.
4.0 Requirements

There are two main deliverables:

(1) Segmented ETDs by annotating the chapter boundaries (100 ETDs)
(2) Summaries for chapters of ETDs - a webpage that accepts chapter text and generates summaries, and an API to call for generating a summary

The segmented ETDs are ETDs that we manually split into separate PDFs using tools like PDF2GO [16]. The segments were mainly starting at chapter boundaries; however there were also other boundaries we had to keep in mind. Everything before the first chapter of content was its own segment, the “front” segment, and depending on what the document had at the end an appendix/resources ending segment was identified as well.

Another requirement is to create scripts for data extraction and summarization using existing Python libraries and language models. They need to be able to extract content of the ETDs correctly, as we specify. There are a few main elements we wish to ignore, as they are the most common parts of ETDs that take away from the main content. Figure 2 shows what we consider a complete and comprehensive list of elements that we wish to remove through the data extraction. These elements are useful and important parts of the document; however we only want the main body of text for summarization. For our purposes, we will often refer to these elements as “noise”, since for summarization these elements are unnecessarily cluttering the data we actually need to use. After removing this noise, we are then able to completely extract the other data in the PDFs.

Excluded Elements in Data Extraction:
- Page Numbers
- Equations
- Images
- Figures
- Citations
- Chapter Headers
- Page Headers
- Footers
- Tables

Figure 2: List of Elements Excluded in Document for Data Extraction
Given the ability to extract data completely and correctly, we need to use language models to create summaries for some extracted chunks of data (representative of a chapter of information) in a way that is comprehensive and abstractive. Creating scripts that manage to do all this so far is the backend portion of this project.

The final stage and requirement of this project is to create a webpage that accepts chapters and outputs summaries. Creating this website to interact with our summarization code is the frontend portion of our project. The intention is that given a chapter we can create the summary and output it in a usable manner.
5.0 Design

5.1 High Level Backend Design

The team’s initial focus was on finding and manually segmenting ETDs. Here we had to meet a few requirements. For example, the ETDs had to be born digital and they had to span a variety of fields and interdisciplinary work to ensure diversity in our ETDs. Despite formatting and organization, we tried to segment the chapters intuitively to ensure that the segmentation was as similar as possible among all the ETDs. Figure 3 illustrates these efforts on the left side, and identifies ongoing and future efforts on the right side.

The next part stage of the pipeline is the data extraction of the useful text of the ETD chapter summaries. Here we use the Object Detection Code by Aman Ahuja and others [1] to get the wanted text as an output.

We take the extracted text and run it through a summarizer script which will output a summary of the given chapter data in a usable way. The summarizer script will utilize models from HuggingFace, “the AI community building the future” [8]. The plan is to create 4 summarization models which the user can select from to summarize an ETD.
5.2 Frontend Design

Our frontend design mostly centered around improving and tailoring the initial design that our team inherited from a previous graduate group that worked on this project. While the design was fairly comprehensive, and even had extended functionality past the scope of this project, it was still rough around the edges from a UI perspective. After half of our team shifted focus to work on the frontend, we created a number of wireframes that outlined the changes and additions we wanted to make to the design. The major changes proposed were various alterations to the Experimenter page, the Curator page, and the addition of a summary database page.

Firstly, for the experimenter page, the proposed design looked to get rid of the original website’s grid layout. In the original design, all of the left widgets were labels, but with the shadows added around the borders of the dividers, they appear to be buttons. The design was not very intuitive, so we simplified the page to make it look cleaner. In our design, the drag and drop box for file uploads would take up a majority of the space. The labels from the original design would be overlaid on the drop menus when nothing has been selected. Also, per our client’s request, we limited the number of pages that an uploaded file could have, and added an adaptive display that advertised the page limit, indicating whether an uploaded file is within the file limit.

The original design and our wireframes for the proposed changes can be seen in Figures 4-7.
Figure 5: The Proposed Experimenter Page Design

Figure 6: The Proposed Experimenter Page After a File Within Page Limit is Uploaded
Next, our design for the Curator page (see Figures 8 and 9) aimed to add a similar drag and drop widget for file upload similar to the one added to the Experimenter page. We designed it this way to get more consistency between the two pages and to fill some of the whitespace present on the original Curator page. Another feature that we thought to include was the disabling of the submit button on both the Curator and Experimenter pages. When observing the current deployment of the site, we found a bug where the site will permanently buffer if either of the buttons are pressed when no file is submitted. To avoid this situation, the button should only become pressable when there is a file uploaded.
Lastly, our proposed design would add a new page to the site that would allow for searching through a database of summaries that have already been processed by our system. This page would allow the user to search for keywords that may appear in the title, abstract, or
summary of an ETD. The page would also allow for the user to apply filters on their searches. These filters would include things like which algorithm was used to generate the summary or the discipline that the ETD falls within. The basic design of the search page for the summaries can be seen in Figures 10 and 11.

![Summary Database Page](image_url)

*Figure 10: The Proposed Design for the Summary Database Page*
Figure 11: The Proposed Design for Displaying Results on Summary Database Page

Our design for the frontend also includes various other minor changes and additions to the site; see Figures 12 and 13. These include the addition of a second logout button at the top right for easy access, customized side headers depending on the page of the site being displayed, rearranging the sidebar navigation to be more consistent, and the ability to add a profile picture.
Figure 12: Illustration of Minor Frontend Changes Proposed

Inclusion of 2nd logout button

Ability to upload profile picture

Figure 13: A Comparison Between the Original Sidebar (left) and the Proposed Sidebar (right)
For the sidebar navigation, our team found the original layout confusing. The Search Experiment page button being between the About page and logout buttons seemed counterintuitive, so we moved it below the Search page. The new Summary Database was then put below the Search Experiments page, to give the navigation a more consistent look. Moreover, the ‘Search’ page was disambiguated in our design to the ‘Search ETDs’ page. We removed the Settings page from navigation because it does not have a physical link or design to it within the frontend code, and at this time, we saw no use for it.
6.0 Implementation

Our final deliverable is a web page that accepts chapter text and generates summaries, and an API for the summary generation. This requires both a backend and frontend portion. For more details about the backend, see Figure 14 and the next subsection.

6.1 Backend Implementation

![Backend Pipeline Diagram](image)

*Figure 14: Backend Pipeline Diagram with Libraries Used at Each Stage for Data Extraction*

This diagram shows the data extraction process. We used the Object Detection Code [1], which utilized Yolov7 [28] to generate bounding box information for the ETD PDFs in the form of .csv files. The bounding box output includes page coordinates for all of the elements commonly found on a research document, like titles, subtitles, paragraphs, images, etc., which we need for the next step. That information was then ingested into a python script as a Pandas dataframe, and used as extraction boundaries for our text extractor to avoid irrelevant information for summarization, including but not limited to equations, tables and references. We
then extracted all the relevant text information into a .txt file for every PDF given using PDFPlumber’s [17] text extraction capabilities.

After the data extraction process, we created multiple summarization model scripts that would take the extracted data text and output a summary of the given text. The summarization models were created based on existing models in HuggingFace. We implemented the models using the HuggingFace Transformers library [24] to use the following pretrained models: BART [22], T5 [10], BigBirdPegasus [23], and Longformer Encoder Decoder (LED) [14].

There were two main ways the models were obtained: using the pipeline function [7] or using the specific model’s conditional generation function from the transformers library [24]. For example, if we’re using the T5 model, then we create the model by using pipeline(“summarization”, “t5-[size]”) or T5ForConditionalGeneration.from_pretrained(“t5-[size]”) [10]. After collecting all the files we need to summarize we pass them through the main summarizing part of our script. Here we take in a certain chunk of data, create a summary, output it, and repeat for the entirety of a single chapter .txt file. This is important because most models cannot handle processing a lot of words at once. At the very least, these models cannot take in an entire chapter of an ETD and process it at once to give a cohesive summary. Most of the models we used can only process about 1024 to 4096 tokens/words at a time, and this is common among most NLP processors.

For the scripts that used conditional generation to create the models (T5 and BigBirdPegasus), we used tokenizers to create tokens based on the input text, run a set of tokens through the summarizer model, decode the tokens post summarization, and write them to summary .txt files. For testing purposes we used the “t5-small” model, but in the actual scripts that we used to get all our summaries we used the “t5-large” model. The T5 model [10] can take in up to 1024 tokens at a time for processing, and we used the T5Tokenizer. The BigBirdPegasus model was based on the specific model “google/bigbird-pegasus-large-arxiv” [23]. It can take in up to 4096 tokens at a time for processing, and we used the AutoTokenizer since we were having issues with the BigBirdPegasusTokenizer.

For the scripts that used a pipeline [7] to create the model (LED and BART), we decided to forego tokenization to see how this would affect the output summaries. The process of what
happens is the same as above, just without the tokenization encoding and decoding. The BART script is based on the "facebook/bart-large-cnn" model and it can process 1024 tokens at a time [22]. The LED model is based on “pszemraj/led-base-book-summary” and it can process 16384 tokens at a time. For each of the scripts, the text extraction files were processed in chunks of the respective size that the model would allow. For example, the BART summarizer would process a text file with 10,000 tokens in around 10 chunks, while the LED based summarizer is able to process that in one chunk. Currently “chunks” are simply the maximum number of tokens a model can take in at a time. So if a model can process 1024 tokens at a time, then every chunk is 1024 tokens, with the exception of the last chunk which will have some number less than or equal to 1024 (or the model’s respective maximum) to account for the last bit of the input text. At this time, we do not make chunks with respect to sentence boundaries. LED and BART share characteristics that made the implementation of the two quite easy. The LED input actually uses the same positional encoding as BART, copied 16 times, so the format of the input was identical, just longer in length.

One model that was unsuccessful in competently producing summaries was GPT2. Using the medium version from HuggingFace, we attempted to generate summaries using the “TL;DR” tag feature at the end of the text prompt. However, most of the summaries were around the same length as the input, making it more of a “rewording” than a summary. Since GPT2 is an auto-regressive model, meaning it is finetuned for prediction of following tokens, it proved to not be suited well for advanced summarization tasks.

We also implemented multiprocessing in our data extraction scripts and summarization scripts using the Pool’s map function in Python’s multiprocessing library [13] with something similar to pool.map(summarize, glob_list). In this case, “summarize” is the function where all of the files in “glob_list” are called to complete the summarization process.

6.2 Frontend Implementation

After our design phase for the frontend, we began to append to the existing frontend application that already had much of our required interface completed. The code was developed
using the React.js library. The website accepts ETD chapter PDFs, allows the user to make a decision on which summarization model will be used from a predefined list, and generates chapter summary text. At this time, after discussion with our client, we decided to cease working on our frontend implementation despite not accomplishing everything we hoped to. This decision came after an error with integrating our database halted our progress. This will be discussed in further detail later on in the report. The work that has been completed for the website is primarily alterations of the legacy code that we inherited from a previous project group to better suit the needs of the client.

We ultimately had to deprioritize some of the cosmetic changes that were proposed for the design, as well, in an attempt to get the website functioning as intended. The changes that have currently been made include the implementation of the page limit, the disabling of the buttons on the Curator and Experimenter page until a file is uploaded, the alteration of the sidebar navigation according to our design, and fixing of the typos in the documentation. Hosting our website was also something deprioritized by request of our client, so all of the following screenshots are from local hosting only.

The addition of page limits on the Curator and Experimenter page was done through the use of the JavaScript library known as PDF-LIB [2]. At a high level, the implementation uses a file reader and an array buffer to read the uploaded file and checks the number of pages that the uploaded PDF contains. Once a file is uploaded on either the Curator or Experimenter page, the submit button on either page will be enabled, allowing the user to attempt to process their file. However, if the page is not within the page limit, the file will not be processed and an alert will pop up on the user’s screen reminding them of the page limit, prompting them to upload a smaller file. If the file is within the limit, the system will process the uploaded file on the respective operation. While using alerts was not a part of our original design, it was one of the cosmetic tradeoffs that has been sacrificed to attempt to finish the application’s core functionality. It was also the method that the previous group used to display information, so we looked to maintain consistency, as well.
Figure 15: The Alert upon Successful Upload for the Curator Page

Figure 16: The Alert for an Upload that Exceeds the Page Limit on the Curator Page
Along with these changes, the Experimenter page also had the widget for choosing a classification model removed. This was done at the request of the client, who said because it is not within the scope of our project, that it would be better to have the drop down menu removed from the page entirely. Moreover, both the Curator and Experimenter pages now have a header that specifies the page limit to the user. These changes can be seen in Figure 18, which also displays what the disabled submit button looks like when no file is uploaded to either the Curator or Experimenter page.

![Experiment page](image)

*Figure 18: The Redesigned Experimenter Page*
For the purposes of our Summary Database, we used the PostgreSQL database management system to store data ingested and output by the summarization pipeline running in the backend. In particular, the database consists of three tables: Chapters, ETD, and Summary. The chapters table contains data about the ID, the ID of the ETD it’s associated with, the chapter number, and the local path. The ETD table contains more detailed information pertaining to the overall ETD document – the ID, title, author, advisor, year published, the abstract, university, degree, URL, department, discipline, and language. Finally, the summary table contains data pertaining to the output of the summarization pipeline, in particular, the system ID, the ID of the ETD associated, the chapter number, the actual summarized text, the algorithm/model used to summarize, and the local path.

<table>
<thead>
<tr>
<th>ID</th>
<th>ETD_ID</th>
<th>Chapter_no</th>
<th>Summarization_text</th>
<th>Algo_used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62903</td>
<td>1</td>
<td>From the earliest times when people started…</td>
<td>BART</td>
</tr>
<tr>
<td>2</td>
<td>63571</td>
<td>1</td>
<td>Cancer is a disease in which a group of cells…</td>
<td>BART</td>
</tr>
<tr>
<td>3</td>
<td>64783</td>
<td>2</td>
<td>A new high power MPD thruster M PDT…</td>
<td>BART</td>
</tr>
</tbody>
</table>

Table 1: Sample data from the “Summary” table of the database

The summary database is complete with nearly 20 of the summarized outputs from the summarization pipeline. Although this was intended to work together with a separate “view summaries” page in our frontend, this work was unable to be completed due to dependency issues and complication of the code written by previous teams. This database was intended to provide users with a central area where they can view summaries outputted by the pipeline in addition with details like word count and the algorithm use.
7.0 Evaluation

The project went unfinalized due to the frontend issues and time constraints. This evaluation will solely focus on the work that we have completed, but will take into account the missing pieces of our project.

The team is testing the scripts we have created for data extraction using the segmented ETDs we created earlier and object detection code we were given by Aman Ahuja et al [1]. We are close to the level of data extraction we would like, but still facing a few problems. But we are able to use the given resources to visually and iteratively test our data extraction scripts.

The summarization model scripts create logically correct summaries based on the given inputs. We have gone through and seen how well these summaries actually summarize the given inputs on a general level. In a more rigorous evaluation, we need to make sure that they are not too long and make sense grammatically and are abstractive. We noticed that some of the models (this was especially apparent in the T5 model) had trouble outputting summaries that met the minimum word count we gave them, causing them to repeat words over and over again. This means we can update the script to perhaps have a smaller word minimum for the summaries. The T5 script outputs the summaries with a “<n>” for every time it has a period or an unknown character. This issue could not be fixed with additions to the decoder portion of the code, so finding a way, like post-processing, to fix that issue, is necessary. We might need to use a different model in the case that this is an inherent issue as well.

The completed frontend work primarily addressed the concerns that we had with the previous design as well as adds and removes certain functionality that made the webpage more suitable for our application. In particular, our webpage removes redundant pages which were not relevant to our application, and adds application-specific functionality like a dedicated page to view all summaries in one place, as well as a page limiter. Previously, we determined the evaluation of the frontend should be based on its ease of use, interoperability with our backend pipeline, and overall functionality pertaining to the frontend UI elements.

We believe that our frontend has strong UI improvements to make the website look cleaner and minimize user errors when navigating through the site. Unfortunately, these could
not be fully tested with a user base due to the web interface being left incomplete. However, the changes are notable between the versions and we feel that the ease of use and UI elements would’ve scored well had they been brought to a testing group.

Unfortunately, since our frontend and backend were not properly synchronized, this leaves our interoperability between the two elements to be completely untested. For these reasons, we would need to evaluate this aspect of our frontend work to be a failure. Despite this, we believe that the work we have completed is a step in the right direction and would supply future developers with a good amount of progress in the right direction.

Overall, while our work on the frontend aspect of this project led to some additional features and improvements of a previous project, the result was underwhelming. This is the result of a number of setbacks and mistakes on the part of our team. Time is the leading factor in why we were not able to complete this deliverable, and this could have been remedied had our team split into backend and frontend sub teams prior to mid-March.
8.0 User Manual

Since our team was unable to complete the frontend, we do not have a completed product to demo. Our theoretical design for the frontend can be seen in various figures in Section 5 and Section 6.

To reiterate, the Experimenter page, as seen in Figure 18, was designed to be able to allow users to upload PDFs containing separated chapters of ETDs to be summarized according to a certain summarization model. These PDFs would be required to be within the page limit, as displayed in Figures 15 and 16. This summary, generated by our backend pipeline, would be output to the user and viewable within the Summary Database; wireframes of this page can be seen in Figure 10 and Figure 11. The Summary Database page would’ve allowed for various queries of the database information, displayed in Table 1, to parse through it and allow the user to find the information that they were looking for. We then would’ve liked the results of these queries to be easily filtered and exported, for ease of use for the user.

8.1 Use cases / Tasks Supported

The main users we are expecting to utilize our system are people reading papers for information. These users (see Figure 19) will likely be students or other academics looking to learn information from ETDs, but in the case of our system, we break these into four subcategories. These subcategories are: Curators, Experimenters, Authors, and those interested in managing the collection of ETDs after they’ve been segmented and summarized – we will denote this persona class as Managers.

The main use case of our system would be by the Curator. The Curator would be the person with a collection of ETDs, who is looking for summaries for the individual chapters of the ETDs within their collection. The Curator would create requests for summarization by specifying a set of one or more ETDs, or by inputting the desired chapter(s) of their text into our web interface. As previously discussed, this text would be parsed, put through frequency analysis, and then output a summary that is deemed suitable. The Curator would then be able to view the summary from within our database or export the summary for their own records. A high level breakdown of these tasks can be seen in Figure 19.
The second type of user of our system would be Experimenters. In this case, these users would be those who are looking to refine the system for segmentation and summarization. This could come in the form of another team working on this project in the future. These users may also be looking to give feedback on the summaries being produced by the system.

Our third type of user that we are designing our system for is authors or researchers. These are the users who are looking to increase the exposure of their ETDs. Our system will help them achieve this goal by making their papers more accessible by increasing the amount of metadata that is available for their ETD. Our system will allow for these summaries to be exported, so this will provide authors with derived chapter metadata that can be incorporated into search engines. By doing this, readers will find ETDs that more closely match their searches, which will help the exposure of cross-discipline ETDs, thus potentially increasing the exposure of an author’s work. This specific use case can be seen in Figure 20.
Figure 20: Diagram Displaying Task Breakdown for Use Case for Authors
9.0 Developer Manual

9.1 Figures

As seen in Figure 21, all of our work is in the dev branch in our GitLab repository [9]. The bounding_box directory has all the code and information we need to run the Object Detection code by Aman Ahuja and others [1]. The chapterinfo directory has two .txt files that describe the data extracted from the STEM and non-STEM ETDs; it contains information like the number of words that each chapter in each ETD has, and the average number of words in each chapter in each ETD in a non-STEM discipline. The data directory contains the extracted text from the STEM ETDs. The generatedSummaries directory contains multiple folders
containing the resulting summaries of each summarization model we made, as seen in Figure 22. The nonSTEM_data directory contains the extracted text from the non-STEM ETDs. The scripts folder contains all the separate scripts we used for the data extraction pipeline and the summarization pipeline.

<table>
<thead>
<tr>
<th>Name</th>
<th>Last commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td></td>
</tr>
<tr>
<td>BART-large-summaries</td>
<td>adding summaries</td>
</tr>
<tr>
<td>BigBIRD-STEM</td>
<td>II bigbird summaries</td>
</tr>
<tr>
<td>BigBIRD-nonSTEM</td>
<td>II bigbird summaries</td>
</tr>
<tr>
<td>LED,STEM</td>
<td>LED stem summaries</td>
</tr>
<tr>
<td>LED,nonSTEM</td>
<td>added LED non stem</td>
</tr>
<tr>
<td>summary,STEM,BART</td>
<td>adding all stem summaries</td>
</tr>
<tr>
<td>summary,nonSTEM,BART</td>
<td>non STEM summaries BART</td>
</tr>
<tr>
<td>t5-STEM</td>
<td>adding t5 stem</td>
</tr>
<tr>
<td>t5-nonSTEM</td>
<td>t5 non stem</td>
</tr>
<tr>
<td>.DS_Store</td>
<td>non STEM summaries BART</td>
</tr>
<tr>
<td>.gitkeep</td>
<td>Add new directory to record the</td>
</tr>
</tbody>
</table>

Figure 22: Generated Summaries directory in GitLab Repository

Figure 22, as mentioned above, is a screenshot of the generatedSummaries directory which contains a bunch of directories which sort the generated summaries of each summarization model based on STEM and non-STEM ETDs.
```python
hf_name = 'pszemraj/led-base-book-summary'
summarizer = pipeline(
    "summarization",
    hf_name,
    device=0 if torch.cuda.is_available() else -1,
)
```

**Figure 23: LED Summarization Model**

Figure 23 in a screenshot of the script that utilizes the LED summarization model. The “hf_name” variable can be modified to try different summarization models from HuggingFace.

```
CHUNK_SIZE = 16384
```

**Figure 24: CHUNK_SIZE Variable**

```python
def summarize(globlist):
    chunk_num = 0
    out_filename = file_name + "_summary.txt"
    out_file = open(out_filename, "w", encoding="utf-8")
    fstat = os.stat(file_name)
    f = open(file_name, encoding='utf-8')
    bytes_left = fstat.st_size
    while bytes_left > 0:
        print('Processing chunk ' + str(chunk_num))
        chunk_num += 1

        prompt = f.read(CHUNK_SIZE)
        prompt_length = len(prompt)
        if prompt_length < 1000:
            bytes_left -= CHUNK_SIZE
            continue # Not large enough chunk to be worth summarizing

        (min_length, max_length) = calc_length(prompt_length)
        out_file.write(create_response(prompt, min_length, max_length))
        bytes_left -= CHUNK_SIZE
```

**Figure 25: Summarization and Chunking Logic of LED Summarization Script**
Figure 24 displays the CHUNK_SIZE variable. This is utilized in every script and it is modified accordingly to accommodate maximum input length for chosen model. For example, models that use the Longform Encoder Decoder base, like the novel summarization model shown in Figure 24, can handle 16,384 input tokens. Other models like BART [22] can only handle 1024 tokens at a time. Therefore the CHUNK_SIZE variable is set to 1024 in the BART summarization model script. This is implemented within the scripts, and not something that a user will have to manually change in the code to use.

Figure 25 shows our logic for the LED summarizer script. The other models more or less follow the same logic with minor changes to account for small changes between models. One change for example is the varying CHUNK_SIZE depending on the maximum number of tokens a model can handle/process at a time. We went into more detail about the differences between the models’ scripts in Section 6, considering differences in how we import models and the use of tokenizers.

9.2 Inventory

- **Frontend**: [https://code.vt.edu/bipashabanerjee/cs-5604-frontend](https://code.vt.edu/bipashabanerjee/cs-5604-frontend)
  - Frontend Source Code
- **Backend**: [https://code.vt.edu/bipashabanerjee/cs4624_chaptersummarization](https://code.vt.edu/bipashabanerjee/cs4624_chaptersummarization)
  - Backend Source Code
- **Data Collection**: See Appendix B and Appendix C
  - Appendix B contains notes about the STEM ETDs, the ETD IDs, how we segmented them, and any issues we had with the data collection and cleaning phase.
  - Appendix C contains notes about the non-STEM ETDs, the ETD IDs, how we segmented them, and any issues we had with the data collection and cleaning phase.
10.0 Lessons Learned

10.1 Timeline/Schedule

Figure 26 shows the team’s first timeline created in the initial stages of the project.

![Timeline Diagram]

*Figure 26: Rough Overview of Timeline and Current Progress*

The team first met in mid-January to get a better understanding of the project and the expectations of this project. After understanding the project on an intermediate level, we created the timeline in Figure 26. This timeline describes our general biweekly goals and milestones. At this point we had a good understanding of the initial data preparation and therefore had more details for the first two milestones. In the next two weeks, the team gained an in-depth
understanding of the project; we used the resource Ally to create OKRs (Objectives and Key Results) and used these OKRs to create the timeline shown in Figure 27.

<table>
<thead>
<tr>
<th>January</th>
<th>February</th>
<th>February - March</th>
<th>March</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chapter Preparation</strong></td>
<td><strong>Research and Design</strong></td>
<td><strong>Design and Implementation</strong></td>
<td><strong>Completed Website</strong></td>
</tr>
<tr>
<td>Finalize Segmenting ETDs</td>
<td>Python Libraries Research</td>
<td>Implementation of NLP code base</td>
<td>Optimized NLP pipeline after</td>
</tr>
<tr>
<td>Complete Data Cleaning</td>
<td>Data Extraction</td>
<td>Design for backend+frontend of website</td>
<td>iterative improvements</td>
</tr>
<tr>
<td>Test with NLP Pipeline</td>
<td></td>
<td>Completed frontend design</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 27: Timeline Containing Technical Milestones*

This timeline is much more compact but it contains clarified technical milestones that we didn’t initially have. In addition to these milestones that we have mapped out, the team has weekly meetings with the client in addition to team check-ins multiple times a week. We have mostly kept up with the timeline, but were slightly behind regarding having the frontend and backend connected by the end of March.

10.2 Problems

The first issue that we faced was in the data collection phase. We had to manually segment the given STEM ETDs, and then proceed to find approximately 50 more non-STEM ETDs. The biggest issues here were finding ETDs that our client had access to download and ETDs that were born digital. We needed ETDs that were born digital to prevent Optical Character Recognition (OCR) issues when dealing with scanned documents. For example, if an ETD was uploaded as a PDF that consisted of scanned images, that we proceeded to try to scan/parse this information again for text extraction as part of our NLP pipeline, we run into
OCR issues as there are limitations to scanning technology resulting in inaccurate information parsing and conversion.

Once we had found and started segmenting ETDs, we ran into issues with formatting and organization. For the most part, we had no issues with segmenting STEM ETDs, since they all followed a similar organizational and formatted structure. Segmentation was mostly intuitive, until we ran into issues like every chapter having its own appendix and the end of one chapter being on the same page as the starting of another chapter. For such chapters with special cases, we segmented the entire chapter as usual, and notated them in an Excel spreadsheet containing information on each ETD to ensure that the text extraction accounted for those situations by ignoring the irrelevant text.

Upon completion of the data cleaning and collection phase, we started working with the skeleton code to start the data extraction process. After experimenting with the ETDs we had collected, we noticed a lot of issues with our ability to recognize many of the noisy elements. At this point, we had no implementation that found noise, such as page numbers, equations, and citations, but the code was intended to eliminate tables and images via Python library PDFPlumber [17] and bounding boxes. We found that in some cases where ETDs were of poor quality, the data extraction code was not able to pull any information, and returned empty .txt files. This was interesting, as our client ran that particular ETD through a program meant to predict whether a document was born digital, and it predicted that it was. After manual inspection of the ETD, we found there were a lot of errors that made it difficult to read, but by no means did we expect to see no data extracted. One of the uses of the code we used was to get rid of tables. In some ETDs, the program was unable to remove such tables due to the formatting of the tables. We assumed that the Python library we used had a limited understanding of what a table could be, and then we proceeded to make note of ETDs that the code was unable to ignore elements that we expected to be ignored. One lesser issue we found was that the data extraction outputs an equation in its closest ASCII equivalent. For example, an equation with the alpha (α) was output as the lowercase letter “a”. This is not that much of an issue, since at this point we had no code to ignore equations, but it was a reminder to the team that it is very important to remove such noise for the best quality data extraction.
After changing our data extraction pipeline to include more than the original skeleton code and make use of the Object Detection in ETDs program created by Aman Ahuja et al. [1], we ran into more problems due to technical limitations. In order to run the Object Detection in ETDs programs, we each needed to find ways to support running such a large program. The program outputs a variety of information, including PDFs marked up with what elements the program found, and .txt files with information about the bounding boxes based on those found elements. The issue came to the forefront when the program required a GPU which most of the team member’s personal laptops don’t have, so we needed to find workarounds. This required us to use a school server with a GPU to obtain the ObjectDetection output. This required us to use a school server with a GPU to obtain the ObjectDetection output.

After getting the ObjectDetection output, we needed to complete our pipeline. As mentioned before, one of the results of the utilized program was a .txt file with the string representation of all the bounding boxes information related to every object detected. Utilizing this information, we parsed through all of the bounding box information and hoped for a simple integration. Unfortunately, we ran into integration issues due to the different scales of the dimensions of the bounding boxes determined by the Object Detection [1] program’s Python libraries (pdf2image [11]) and our data extraction’s Python library (PDFPlumber [17]).

We have a completed data extraction pipeline, but we ran into this issue where sometimes words wouldn’t be read correctly by the data extraction script. We often saw this issue with letter combinations like “fi”. We concluded that this was due to a lack of space between the letters, causing the extractor to miss it. Additionally, oftentimes the output of the data extraction pipeline was one large chunk of text where all of the extracted text was one big word with no spaces.

The summarization pipeline was supposed to consist of 3-5 summarization models. We originally decided on BART [22], Longformer Encoder Decoder (LED) [14], BigBirdPegasus [23], and GPT2. However, as mentioned before, we ran into issues with the GPT2 model where the “summaries” were often longer than the original text.

All of these models are very large and run best on computers with a lot of computational power. This was difficult for those of us developing the scripts, since we only had our laptops to create and test these scripts on.
As for the frontend implementation, we faced two major challenges. The first had to do with our understanding of the code base that we inherited. Hardly any of the code was documented, and it seemed to have passed many hands over the past year, so it was difficult to understand the function of certain elements. The return values of the JavaScript files are different between the different pages. None of our group members have strong development skills in JavaScript, so understanding the differences in the pages caused setbacks when it came to integrating features. This is because reusing our solutions often conflicted with the Grid API that was used by the previous developers, leading to many features being unfinished due to time constraints.

The second major issue that we encountered was an integration error when attempting to deploy the website locally whilst also trying to set up a connection to our Summary Database. Despite having what we believed to be the correct node modules installed correctly, when attempting to add a connection to PostgreSQL, our code would not compile correctly. The error message that appeared can be seen in Figure 28. This is where the group’s inexperience with JavaScript started to show, as we relied heavily on external sources on the Internet to help remedy the error. However, none of the solutions attempted fixed the source of the error.

Failed to compile

```bash
/node_modules/pg/lib/native/client.js
Module not found: Can't resolve 'pg-native' in 'C:\Users\grill\OneDrive\Documents\Spring 2023\Multimedia Capstone\Frontend\chapter-summarization-frontend\node_modules\pg\lib\native'
```

This error occurred during the build time and cannot be dismissed.

**Figure 28: Integration Error When Setting Up Connection to Summary Database**

10.3 Solutions

To combat the first issue we encountered with the potential OCR issues, we made sure to collect a plethora of ETDs that were made around 2010 and after. This way they are more likely to be born digital and not be a PDF consisting of scanned-in images. We also utilized a machine learning program that our client Bipasha has access to which predicted whether a document was born digital as a PDF, or was an image (scanned document).
The next issue we faced during the data collection phase was with the manual segmentation of the ETDs. We tried to follow a segmentation pattern that grouped all the front parts and end parts together respectively and each chapter separately. As stated, we only ran into these problems occasionally. When we did we approached this with the idea that our pattern has to be intuitive, so we chose a pattern and stuck with it for all the ETDs that we segmented.

In order to overcome the limited ability of PDFPlumber [17] specific to ignoring tables as well as its inability to ignore other noise, we made use of a program, Object Detection in ETDs [1]. This program takes in chapters, and outputs a visual representation of the chapter including PDFs marked up with what elements the program found and a .txt file with information about the bounding boxes around those elements. Using this program we were able to easily identify all the noise we needed to remove in addition to other elements. This way we will be able to integrate the information about the bounding boxes into our current data extraction pipeline to easily account for all of the noise in a more accurate manner than could our skeleton code which only utilized PDFPlumber [17].

To combat the technical limitations we faced when trying to run the Object Detection in ETDs program, we were aided by Bipasha and used a server with a GPU. We then took the results and added them to our shared drive and kept track of all the information in a way that was easily accessible to all the team members. At this point it was much easier to integrate the information from the other program into our pipeline.

The dimension issue is one that we are currently still working through, but we are close to resolving. We had to do a lot of R&D about the libraries used in all the programs to see what dimensions they were using. This was a more difficult task than it sounds like because the documentation pages for both PDFPlumber [17] and pdf2image [11] (the two main libraries that gave us the information) were sparse and we needed to do more research to find hypothetical values. After going through some hypothetical scaling values, we ran a PDF where an image was the entire page to see what dimensions pdf2image [11] was utilizing. Then, we made the proper changes so that it accurately reflects the dimension of PDFPlumber [17]. This mostly fixed the issue. However, there’s still a minor issue with the x-coordinates of the bounding boxes being slightly off in size, which causes some text near the side margins of the PDF pages to be cut off.
Unfortunately, we did not find a solution to the issue where the libraries were not able to read the script as it is an OCR issue more than anything we could solve. However, for the issue where all the outputted text was one large word, we used the WordNinja [19] library. This library is known for taking in a chunk of text and splitting it up into words. We used this to create more readable extracted text. Unfortunately, we also ran into some issues where Word Ninja didn’t recognize some more niche words in STEM ETDs. When we weighed the pros and cons of using WordNinja [19], we decided to keep use of the library as it outputs more cohesive data. However, this may be subject to change in future work, as we mention in the next section.

In an effort to maintain the goal of 4 different summarization models, we pivoted from using the GPT2 model to using the T5 model instead. We specifically used the “t5-large” model that is the medium sized model of the available sizes: t5-small, t5-base, t5-large, t5-3b, t5-11b [10].

In order to combat the issue of creating such scripts on our laptops, we had to do research into how we could run these large models. We found that there are ways we can elect to use CPU and vice versa. We added that to our scripts when we tested them. This was still a very slow process and it would require hours to run all of our chapter ETD files through these scripts. We combatted this issue by testing in small batches and implementing multiprocessing. Once we had the scripts running with some success on our small test batches, we handed the scripts over to our client, who had access to more powerful servers and could run such a large number of files through the scripts that utilized advanced summarization models.

To address the difficulties in integrating our Summary Database with our website, we attempted various solutions that we found through research. These included downloading the pg-native library globally, adding PostgreSQL to our local paths, updating all node_modules, deleting and re-downloading the node modules, and manually and automatically auditing the code for conflicts. However, none of these solutions worked. Eventually, the group decided that the only solution would be to recreate the site from scratch for our specific purpose. However, after discussing the matter with our client, we realized that we did not have enough time to create a site from scratch. Therefore, we decided to halt the progress at this time.
If our group had developed the web interface from scratch, we would have had more control over the site's structure and the APIs used, instead of trying to tweak the legacy code to our needs. Attempting to reuse the legacy code created several issues with our understanding that could have been avoided if our group had coded and documented the website in our own style, and only focused on the smaller scope of our project.

10.4 Future work

The backend portion consists of data extraction and summarization pipelines. While we have implemented a version that works well enough for the completion of the project, they can be improved. We have contemplated the use of libraries like WordNinja [19] to see if it would be better for the data extraction process. There are good and bad things in our data extraction process, but perhaps there are other ways to create a more robust and accepting data extraction pipeline. In the future, this project will be greatly improved if the data extraction pipeline is improved so that the raw data that is summarized can be more accurate relative to the information in the original ETD chapter.

Similar to how we can improve the data extraction, the summarization script can always be improved. We based our summarization models on highly developed models, but there is always room for improvement in terms of how we deal with the implementation of the models. For example, we may have decided to use a tokenizer encoder and decoder for one model, but experimenting without using it may produce better results. We made these models with suggested minimum and maximum word counts for the resulting summaries; perhaps changing these boundaries will produce better results. Experimenting with the padding and truncating aspects of the models could yield better summaries as well. As mentioned in Section 6.1 Backend Implementation, we currently read input data in chunks. These chunks consist of the maximum number of tokens each model can process until the end where it reads the remaining tokens left in the input data (which are usually less than the maximum number of tokens); it may be beneficial to modify these chunks to make sure they respect sentence boundaries, so that the resulting summaries are more grammatically accurate. It is small things in our summarizer
scripts like the things mentioned above that could be changed to work towards a better overall program.

Due to our web interface going unfinished, most of our cosmetic ideas are things that we’d like to see integrated into the future. That is to say, we’d like to see the website take on a similar looking interface to the ones in wireframes that we created. We feel this design is cleaner and more professional looking than the current design. Secondly, the integration of a summary database could be accessible to users through the web interface, similar to how it is described in Section 3.3. Lastly, we recommend a way for users to give developers feedback to allow for continued improvement to the design of the website. For this feature, both frontend and backend support would be necessary. This way the developers can better improve the UI/UX aspect of the website in ways that they may not have initially thought of, and make a better experience for those attempting to use the site.
11.0 Acknowledgements

We would like to thank Bipasha Banarjee for introducing us to the project, leading us throughout the semester, and helping us develop a website that will have global effects for the ETD community.

Bipasha Banerjee - bipashabanerjee@vt.edu

We would like to thank Aman Ahuja, Alan Devera, and others who made the object detection code which contributed to a major part of our data extraction process.

Aman Ahuja - aahuja@vt.edu
Alan Devera - alandevera1@vt.edu

We would like to thank the CS5604 Fall 2022 team guided by Dhanush Dinesh involved in creating the base frontend for our project.

Dhanush Dinesh - dhanushnd@vt.edu

Finally, we’d like to thank Dr. Edward A. Fox for advising throughout this project, and the project that developed the object detection in ETDs.

Dr. Edward A. Fox - fox@vt.edu
12.0 References


13.0 Appendices

13.1 Appendix A: Methodology

This appendix explains how our software is divided based on a variety of tasks and subtasks that interact with each other, and implementation details including the libraries used. See details in Figures 29-31 and Table 2.

QUESTIONS

Q1: Please list and describe the goals of each of the types of users that your system needs to support. For example:

- If you are doing analysis of a dataset, one type of user’s goal is to have the set of information that they are looking to infer or derive.
- If you are building an app or a website or something with an interface, your aim is to have the interface that provides the desired support/functionality, which allows its users to carry out the actions as designed.

- **User: Readers**
  - Looking for better summarization of EDTs with less jargon and supplementary information to aid with comprehending EDTs’ abstracts
  - Looking for better searchability and accessibility of discipline relevant information within multi-discipline EDTs

- **User: Researchers/authors**
  - benefit from more accurate and in-depth summarization as it would allow for more engagement on their papers. Incomplete/misleading abstracts can turn readers away from certain ETDs and lower exposure

- **User: We (the team)**
  - aim to make a webpage that will accept portions ETDs that we parse them and then generate accurate summaries of the given portion of the ETD

Q2: Please break down each goal into units of tasks and subtasks, a combination of which makes up the goal.

- Please remember that tasks are smaller goals.
- Break the goal down into as granular a level as possible such that:
  - a) Tasks don’t overlap with each other, and
If a (sub)task can be broken down further, it should be broken down into subtasks.

- At this stage, each (sub)task should be expressed in a fashion that is user-centric and implementation-agnostic.

For each goal, draw a graph-like figure showing the breakdown of the goal to tasks/subtasks.

To support the goal of examining tweets reporting “outage” during a hurricane, the system needs to support the tasks of:

1) Key phrase matching and sentiment analysis. This task is dependent on the task of 2) doing sentiment analysis on tweets, which is dependent on the system supporting the task of 3) extracting NLP/text-based measures, and so on and so forth.

So, the structure of the graph indicates how tasks are dependent on one another. As a result, can derive a sequence of tasks required to accomplish the goal (as seen above).

What you should turn in is: (a) a set of figures and (b) a short description of each task and subtask.

![Figure 29: User: Readers Breakdown of Goal and Subtasks](image)

**Reader Goals:** Use System’s web interface to summarize a portion of an ETD of interest so that it is more accessible to them. The reader’s second goal would then be to view the summary output by the system.

- Task 1: Create Request for Summary - The users will use our web interface to input a portion of an ETD of interest for summarization.
  - Service 1a: Allow users to input a subsection of an ETD (txt file)
  - Service 1b: Parse subsection and remove irrelevant elements (headers, footers, tables, equations, page numbers, in-text citations/references) and frequently used words (i.e. A, The, etc.)
  - Service 1c: Complete frequency analysis on subsection to find the most commonly used words left in the text
- Service 1d: Use algorithm and summarization rating system to generate accurate summary of subsection

- Task 2: Extract Relevant Information
  - Service 2: Our system will output an accurate summary of the inputted portion of the ETD that the user sent a request for.

**Figure 30: User: Authors/Researchers Breakdown of Goal and Subtasks**

**Author/Researcher Goal:** Increase exposure by increasing accessibility of their papers

- Task 1: Use output from our web interface to increase the amount of metadata available for their paper.
  - Service 1: Allow for easy export of summaries. This will allow them to incorporate derived chapter metadata into search engines to help better fit search results.

Q3: As you develop your solution, for each task, describe the implementation of the service. Please write down implementation-specific information. For example, write down what input file(s) will be required. Which other task is producing that input file? What output file is produced? What are the libraries, functions, and environments that are required? Specify as much detail as you can. The "ID" entries in each table should match an ID in a figure, so it is easy to relate the parts of the figure to the parts of the table (i.e., use ID of "1A" for the first service shown in the figure below).
<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service Name</th>
<th>Input file name(s)</th>
<th>Input file IDs (comma-sep)</th>
<th>Output file name</th>
<th>Output file ID</th>
<th>Libraries; Functions; Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Input subsection</td>
<td>Subsection or entire ETD (user given)</td>
<td>a</td>
<td>Text extraction of input</td>
<td>b</td>
<td>React</td>
</tr>
<tr>
<td>1b</td>
<td>Initial parse of data</td>
<td>Text extraction of input</td>
<td>b</td>
<td>Parsed subsection with removed elements</td>
<td>c</td>
<td>PDFPlumber, Boundary boxing, Glob, timer, open()</td>
</tr>
<tr>
<td>1c</td>
<td>Frequency analysis</td>
<td>Parsed subsection with removed elements</td>
<td>c</td>
<td>Parsed subsection with removed elements and additional metadata</td>
<td>d</td>
<td>NumPy, pandas, glob, Counter(), WordNinja</td>
</tr>
<tr>
<td>1d</td>
<td>Summarization</td>
<td>Parsed subsection with removed elements and additional metadata</td>
<td>d</td>
<td>Thoroughly summarized information of input</td>
<td>e</td>
<td>PyTorch</td>
</tr>
<tr>
<td>2</td>
<td>Output to user</td>
<td>Thoroughly summarized information of input</td>
<td>e</td>
<td>Viewer visible output</td>
<td>f</td>
<td>React</td>
</tr>
<tr>
<td>3</td>
<td>Metadata creation/usage</td>
<td>Increase metadata via exporting summaries</td>
<td>a, e,</td>
<td>Metadata: summaries</td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: Implementation Specific Information for Every Service**

Q4: As you build your solution, When building a system represented as a set of workflows, as shown in the figure below, make a list of workflows covering each goal.
Goal 1 (Use web interface): Workflow 1 = Service 1A + Service 1B + Service 1C

Service 1A: Input subsection of ETD for summarization (PDF of chapter)

Service 1B: Use PDF text parser to convert chapter pdf files to txt format. Use appropriate bounding boxes to avoid figures, images, headers, footers, and other irrelevant data. If any equations are detected in the text, send to an external service to remove these from txt output.

Service 1C: Use WordNinja to split words in case spaces are not detected, then transform data into frequency matrix for use with machine learning.

Service 1D: Run through machine learning summarization transformation to create summary

Goal 2 (Output summary): Workflow 2 = Service 2A + Service 2B + Service 2C

Service 2A: Output summary to user using React frontend

Service 2B: Accept user rating for summary quality (this is not our team’s domain, but is a crucial part of the larger project scope and we must ensure that our two teams create compatible interfaces)

Service 2C: Interpret summary ratings to improve summary quality, flag very bad summaries for investigation

Service 3: Utilize generated summaries as chapter-level metadata for ETDs to enhance searchability and usability.
13.2 Appendix B: First Batch of ETDs

This document is uploaded in our VTechWorks submission with the same title “First Batch of ETDs”. It contains notes about the STEM ETDs, the ETD IDs, how we segmented them, and any issues we had with the data collection and cleaning phase.

13.3 Appendix C: Second Batch of ETDs

This document is uploaded in our VTechWorks submission with the same title “Second Batch of ETDs”. It contains notes about the non-STEM ETDs, the ETD IDs, the ETD departments, how we segmented them, and any issues we had with the data collection and cleaning phase.

It contains notes about how we decided on the ETDs and some notes we had about segmenting and how they worked with our base data extraction methods.