Final Report
CS 4624: Multimedia, Hypertext, and Information Access Capstone
Object Detection & Document Accessibility

Alan Devera, Michael Nader, Zehua Zhang,
Client: Aman Ahuja

May 10, 2023
Instructed by Professor Edward A. Fox

Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
Abstract

Electronic Theses and Dissertations (ETDs) are the primary way that students and professors write down and report their degree research. They allow new minds to understand where that field of study was left off, and how to continue the work that has been left. However, since many of the ETDs uploaded onto the internet are presented via PDF, it’s difficult for users to view these ETDs in an effective manner, especially when you consider potential students with disabilities such as visual impairments. The goal of this project was to extend upon the previous work that has been done to make a Flask-based web application so that we can transform these long documents into something much more readable, user friendly, and accessible via HTML rather than PDF. Also, our goal was to apply an algorithm to the returned bounding boxes that come from the object detection model to make sure that separate paragraphs and references are placed into their own box for correct XML generation on the website. To make the application’s UI usable, we have applied a few changes to improve the experience. We have created the option for users to download the paper via PDF or XML, have a side-bar on the left of the website that contains a dynamic table of contents to jump to whatever part of the paper you select, and have a side-bar view on the right of the website that contains the original PDF so that any errors in our application won’t ruin the user’s understanding. We plan for future contributors to add a dark mode and dyslexic-friendly font. Lots of accessibility features will be added via HTML/CSS/React through improving the UI, but what’s also included is the option to use an on-screen reader. Our project focuses on using NVDA, a popular screen reader, to allow for users with potential visual impairments to be able to listen along to the ETD instead. This was studied thoroughly throughout the course of this project. Finally, for the algorithms side of the project, the focus has been to improve upon the returned bounding boxes from the object detection models to separate paragraph and reference bounding boxes to only include one paragraph or one reference per box. The object detection models do the best they can for the amount of training they’ve received, but errors are still possible. This side of the project focused on fixing those errors from the model to make sure that the XML generation works well and the text is readable on our final application. The algorithms team was able to get a good post-processing algorithm to work for around 90% of the paragraphs in the ETDs that were tested, but were unable to get to the references part of the deliverable. This is left for future collaborators.
# Contents

List of Figures vi

List of Tables viii

1 Overview 1
   1.1 Introduction 1
   1.2 Approach 1
   1.3 Deliverables 2
   1.4 Client 3
   1.5 Team 4

2 Requirements 5
   2.1 Algorithms/Back-End 5
   2.2 UI/Front-End 5
   2.3 Accessibility 6

3 Design 7
   3.1 Algorithms/Back-End 7
   3.2 UI/Front-End 8
   3.3 Accessibility 14

4 Implementation 20
   4.1 Algorithms/Back-End 20
      4.1.1 Paragraph Splitting 20
      4.1.2 XML to HTML Attribute Generation 26
      4.1.3 Text Cleaning 27
      4.1.4 Addressing Case Study Common Issues 27
B.2.1 108017: Investigating Dimensions of Psychopathy in an Adjudicated Adolescent Sample: The Role of Race, Sex and Disruptive Family Processes ........................ 49

B.3 Georgia Tech .......................... 51

B.3.1 41609: Measuring the impacts of hospital nursing floor and patient room layouts on patients’ experience with care in a major teaching hospital ........................................ 51

B.3.2 51993: DEVELOPING AND REFINING LINKS AS A REPRESENTATION OF ORGANIZATIONAL CONNECTION ............. 53

B.4 Texas Tech .......................... 55

B.4.1 261721: The Wide Mouth of Bone: Original Poetry .................. 55

B.5 Virginia Tech .......................... 57

B.5.1 494947: In-Between: Architectural Drawing and Imaginative Knowledge 57
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>All tasks and subtasks that our group will complete</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Object Detection Model adapted from CS 5604 efforts</td>
<td>9</td>
</tr>
<tr>
<td>3.3</td>
<td>Preexisting Object Detection UI</td>
<td>10</td>
</tr>
<tr>
<td>3.4</td>
<td>Preexisting Upload UI</td>
<td>10</td>
</tr>
<tr>
<td>3.5</td>
<td>Preexisting App UI</td>
<td>11</td>
</tr>
<tr>
<td>3.6</td>
<td>Current Object Detection UI</td>
<td>11</td>
</tr>
<tr>
<td>3.7</td>
<td>Current Upload UI</td>
<td>12</td>
</tr>
<tr>
<td>3.8</td>
<td>Final Implementation of Application UI</td>
<td>12</td>
</tr>
<tr>
<td>3.9</td>
<td>Expanded Table of Contents</td>
<td>13</td>
</tr>
<tr>
<td>3.10</td>
<td>Demonstration of PDF Highlighting</td>
<td>14</td>
</tr>
<tr>
<td>3.11</td>
<td>Table of Contents Collapsed and PDF Viewer at Minimum</td>
<td>14</td>
</tr>
<tr>
<td>3.12</td>
<td>Table of Contents Collapsed and PDF Viewer at Maximum</td>
<td>15</td>
</tr>
<tr>
<td>3.13</td>
<td>Table of Contents Present and PDF Viewer at Maximum</td>
<td>15</td>
</tr>
<tr>
<td>3.14</td>
<td>An example of PREP’s tagging before manual alterations</td>
<td>17</td>
</tr>
<tr>
<td>3.15</td>
<td>An example of PREP’s tagging after manual alterations</td>
<td>18</td>
</tr>
<tr>
<td>3.16</td>
<td>The client’s desired system architecture</td>
<td>19</td>
</tr>
<tr>
<td>4.1</td>
<td>Algorithm using CV2 on double spaced paragraphs</td>
<td>21</td>
</tr>
<tr>
<td>4.2</td>
<td>Algorithm using CV2 on bullet points</td>
<td>22</td>
</tr>
<tr>
<td>4.3</td>
<td>Algorithm using PDFminer on paragraph without symbols</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>Algorithm using PDFminer on paragraph containing symbols</td>
<td>23</td>
</tr>
<tr>
<td>4.5</td>
<td>Successful implementation of PDFminer on paragraph containing symbols</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>Latest PDFminer implementation still contains issues with paragraphs containing symbols</td>
<td>24</td>
</tr>
<tr>
<td>4.7</td>
<td>Image containing all bounding boxes required for implementation</td>
<td>25</td>
</tr>
<tr>
<td>4.8</td>
<td>Image Containing all bounding boxes required for implementation</td>
<td>26</td>
</tr>
</tbody>
</table>
List of Tables

A.1 Table indicating each service given and its attributes .......................... 45
Chapter 1

Overview

1.1 Introduction

Graduate students and researchers produce theses and dissertations in order to graduate. Virginia Tech was the first university to require their students to submit these works electronically, starting 1/1/1997. They are known more commonly today as Electronic Theses and Dissertations (ETDs). The problem with these scholarly documents is that they are very long, and parsing them takes time. Because of this, many people do not spend the time to go through these documents. In order to make these documents more accessible, we want to be able to extract all the features of an ETD and store them in a data structure for downstream tasks such as summarizing by chapter and searching figures. In the Spring 2022 semester [1][14], a group of undergraduates led by a grad student sought to solve this problem by first labelling ETDs, training a model to detect common elements found, and creating a data structure (using XML) to store all the extracted elements from ETDs. Building upon this work, during the Fall 2022 semester in CS 5604, Information Storage and Retrieval, Team 3 developed a system that utilizes machine learning-based object detection models to extract elements from long documents. Two major outcomes from this group were the implementation of post-processing rules, such as segmentation by chapter and section, and the creation of a Flask-based web application [6] to display the objects detected from the model. The objective of this project is to expand upon the existing work by upgrading the Flask-based web application, enhancing accessibility for lengthy documents.

1.2 Approach

This project was originally broken up into two separate capstone groups: ObjectDetection and DocAccessibility. For the ObjectDetection team, the project’s main objective was to implement UI changes and add more post-processing rules to those given to us from prior semester’s work. The DocAccessibility’s project objective was to integrate the Flask-based web application with screen readers. Because the two groups would need full communication with each other by using the same existing code-base, the client believed it was best to combine the two capstone projects and split the members into three subgroups: Algorithms/Back-End, UI/Front-End, and Accessibility. Following are their original goals,
that work towards refining all of these aspects of the Flask-based web application:

- **Algorithms/Back-End (Algo)**
  - Goals: Improve segmentation using new models. Break down sections of multiple paragraphs into individual paragraphs. Improve the post-processing of other objects including references.

- **UI/Front-End**

- **Accessibility**
  - Goals: Integrate with screen readers. Conduct some studies of how our tagging compares with those supported by PREP \([4]\).

### 1.3 Deliverables

After talking to our client, we identified the main deliverables that each subgroup would have to complete by the end of the semester. They are:

- **Algorithms/Back-End (Algo)**
  - Implement paragraph and reference separation using tools such as PDFminer.
  - Segment references into their base components (i.e., author name, title, etc.).
  - Add bounding box coordinates, page number, and confidence to each XML tag for downstream tasks that the UI team would use.

- **UI/Front-End**
  - Transition UI to React framework.
  - Add PDF dynamic highlighting to PDF Viewer.
  - Minor changes: accordion table of contents, drag-and-drop upload box, color updates

- **Accessibility**
  - Restructure code available in a GitHub repository.
1.4 Client

The client of this project is Aman Ahuja. Ahuja is a Ph.D. candidate at Virginia Tech. He can be reached at aahuja@vt.edu. His research interests lie in the field of Machine Learning, with emphasis on Document Understanding, Search and Retrieval, and Question-Answering. His Ph.D. research focuses on improving the accessibility of long PDF documents, such as books and dissertations. He also worked as an Applied Scientist Intern at Amazon Search, investigating neural ranking models for product search.
1.5 Team

There are seven members in the group, each with a main role in the team:

- Alan Devera: Leader/contact and Algorithms partner
- Michael Nader: Algorithms presentation leader and Algorithms partner
- Zehua Zhang: UI partner
- Elizabeth Keegan: Accessibility presentation leader and Accessibility partner
- Theodore Gunn: UI report leader and UI partner
- Gabrielle Nguyen: Accessibility report leader and Accessibility partner
Chapter 2

Requirements

2.1 Algorithms/Back-End

The back-end group’s tasks for this project were to use the output of object detection models for a variety of purposes. The existing models for object detection identify and classify elements found in the ETDs, but the previous implementation puts the output directly into the source code. Our client requested us to create intermediary functions to make the output more consistent with the layout of the input PDFs.

The first task given by the client was to fix the separation of paragraphs as they are output by the models. The current models (Detectron2 and Yolov7) have difficulty differentiating between adjacent paragraphs when compared to other objects such as figures. The second task was to format the references and separate them by content. The third task set forth by the client was to use the location of the text boxes to fix the order that text appears in the output.

2.2 UI/Front-End

The purpose of the UI/Front-End is to provide an interface for the user to interact with the data produced by the algorithm. Our client based his vision for the UI on existing websites that showcase ETDs. Previous systems included dynamic tables of contents that include anchor links to allow the user to easily navigate to different parts of the paper.

For our implementation, we used the React framework [10] to enable advanced UI creation. The basic requirements included a collapsible table of contents on the left side of the screen that housed hyperlinks to each chapter, subsection, and figure in the document. We included download buttons on the top right to download the ETD in PDF and XML formats. We implemented a PDF viewer to cross-reference the model’s output with the original document. The PDF viewer is located on the right side of the screen. When the user clicks on an element in the algorithm-produced output, the application highlights the corresponding section of the PDF and automatically scrolls so that the highlighted section is in view in the PDF viewer.
2.3 Accessibility

The Accessibility group is tasked primarily with studying how NVDA [8], a popular screen-reading program, interacts with our system, versus how it interacts with documents tagged by a different system used throughout Virginia Tech called PREP. We were also tasked with restructuring the code to be structured in a more cohesive way, allowing developers to more easily find specific documents, images, and coding modules. Finally, we were tasked with doing a case study of 20-30 theses/dissertations ranging over a wide variety of topics and coming from a large selection of universities, to better understand the benefits and shortcomings of our system.
Chapter 3

Design

As stated above, our group was split into three subgroups to focus work on. Figure 3.1 represents our methodology for this project and the breakdown of tasks amongst us. Details of these tasks are given later in this chapter.

![Diagram of tasks and subtasks]

Figure 3.1: All tasks and subtasks that our group will complete

3.1 Algorithms/Back-End

Figure 3.2 shows the object detection pipeline that is followed when generating an XML file as well as the elements that the Front-End team will use for its web page. This pipeline is adapted from the CS 5604 efforts [3] from the previous semester.
After the object detection model returns the object that contains bounding boxes that it created for each kind of element in the ETD, the next step is for the application to sort through each bounding box (bbox) and place it into the correct part of the XML document to appear on the web page.

Our algorithm was implemented in between those two steps where it intercepts the bounding boxes of any paragraph/references page, applies our algorithm to the original bbox, and separates it into multiple bboxes which will surround only a single paragraph or reference, and then continue to allow the XML to be generated on these new coordinates that are given for new bounding boxes.

3.2 UI/Front-End

Before the user would be able to access the app, he or she must first choose an object detection model and upload a PDF. The app included a page to choose between two object detection models, Detectron2 and Yolov7. Shown in Figure 3.3 is the original page for choosing an object detection model. Finally, the app included a page to upload a file to the selected model. The original UI for this page is shown in Figure 3.4.
To include the features in the requirements, our group designed a website layout that allowed accessibility to several viewing options. The preexisting website included a table of contents as a fixed left sidebar. It also included a hyperlink at the bottom of the page to download the XML file of the ETD. The original app is shown in Figure 3.5.

To update the object detection page, we added a Chicago Maroon top bar with the title “ETD Parser” prominently displayed. We added color and outlines to the model buttons. The current implementation is shown in Figure 3.6. To update the upload page, we added a similar top bar that displayed “File Upload” prominently. We implemented an option to
Figure 3.3: Preexisting Object Detection UI

Figure 3.4: Preexisting Upload UI

drag and drop a file into a box. This button can then be uploaded with the button on the bottom of the box. This implementation is shown in Figure 3.7.
THE ROLE OF SOCIAL INTEGRATION IN THE PERSISTENCE OF AFRICAN AMERICAN MEN IN COLLEGE

University:
UNIVERSITY OF NORTH TEXAS

Degree:

Committee:
APPROVED: Judith A. Adkison, Major Professor V. Barbara Bush, Minor Professor Richard Fossey, Committee Member Nancy Nelson, Chair of the Department of Teacher Education and Administration Jerry R. Thomas, Dean of the College of Education James D. Meermk, Acting Dean of the Toulouse Graduate School

Date:
May 2011

Garrett-Spencer, Jacqueline. The Role of Social Integration in the Persistence of African American Men in College. Doctor of Education (Educational Administration), May 2011, 87 pp., 6 tables, references, 69 titles. This qualitative study addressed the experiences of African American males attending a predominantly White university as undergirded by the social integration aspects of Tinto’s model of academic and social integration. The methodology was case study. Sixteen semi-structured interviews were held with currently enrolled seniors to capture the lived experiences of their reasons for attending college as well as major influences that contributed to their persistence decisions. The results revealed emerging themes of positive and negative family influence, religious beliefs, and a sense of self-efficacy as instrumental factors for the students’ persistence. The level of social integration tended to differ by the age classification (traditional college-going versus non-traditional college student) and by the level of parental education. The components of the social integration model, as developed by Tinto contributed little to the sample’s persistence decisions when compared to the factors suggested during the interviews. Three observations emerged from

Figure 3.5: Preexisting App UI

Figure 3.6: Current Object Detection UI
Our group added to the main page’s design by including a navigation bar at the top of the page that housed a button to collapse the table of contents, and buttons to download the file in PDF or XML form. The group also added a right sidebar to house an expandable PDF viewer, to view the original ETD and the PDF with algorithm-defined bounding boxes. The final implementation is shown in Figure 3.8.

The table of contents is a nested drop-down list of hyperlinks to different parts of the page. On the top level, we have links to each chapter and appendix. On the second level, we have
links to each subsection of each chapter. On the third level, we have links to each figure, which includes images, tables, equations, and algorithms. A demonstration of this is shown in the left side of Figure 3.9.

For the PDF viewer, the group made use of the React PDF Viewer [9], which came with built-in plugins that enabled extra functionality. For our implementation, we used the Toolbar plugin and Highlight plugin. The Toolbar plugin adds buttons to do such things as zoom, view in full screen, and print. The Highlight plugin enables the user to select a portion of the PDF to highlight. We used this plugin to allow the user to select an HTML element on the model-produced output, and then have the PDF viewer automatically highlight the corresponding part of the PDF and jump to page. An example of this is shown in figure 3.10. The topmost paragraph HTML element was clicked, and the PDF viewer highlighted the corresponding paragraph in the PDF viewer.

The group also implemented a feature that would automatically resize the article based on the available space on screen. The table of contents is collapsible on the click of the hamburger icon at the top left of the screen. The PDF viewer is also re-sizeable, able to shift anywhere between 25% and 50% of the user’s screen. The article element, which includes the model-generated content, changes its width to fill in all available space. Examples are shown in Figures 3.11, 3.12, and 3.13.

Figure 3.9: Expanded Table of Contents
3.3 Accessibility

To ensure that the PDFs that are outputted from our Flask application are accessible to the visually impaired, we first needed to conduct some studies on how our program interacted with screen readers. We decided to use the NonVisual Desktop Access (NVDA) screen reader [8], which allows a visually impaired user to navigate through and read what’s on their computer screen. It is compatible with most tagged web pages. NVDA has various
Figure 3.12: Table of Contents Collapsed and PDF Viewer at Maximum

Figure 3.13: Table of Contents Present and PDF Viewer at Maximum

keystrokes that read aloud the onscreen text to the user. For instance, to read aloud the next and previous headings, the keystrokes H and Shift H should be used, respectively. To read the previous/next tagged element/line, the up/down arrow keys should be pressed. To go through the paragraphs, Ctrl+Up and Ctrl+Down should be used. To read aloud individual characters, the user can hit the left arrow key to read the previous character, or the right arrow key to read the next character.

In order to compare how our application worked versus how other widely used applications
worked with screen readers, we compared our system with documents tagged by PREP [4], Virginia Tech’s recommended tagging system. We had different results from using NVDA with our Flask Application versus using NVDA with PREP. When used on our Flask Application, headings and paragraphs are read aloud properly, though lines sometimes started and stopped in incorrect places. Though this is still perfectly readable through the screen reader, anyone visually following along would notice the discrepancy between visual “lines” and audio “lines”. Currently, in our system, tables and figures are recognized as images, so when NVDA reads these elements, it reads the alt-text associated with said element. For the PREP system, the main issue was that there were multiple errors with tagging the headings and paragraphs, which caused issues when trying to read a document using NVDA’s supplied keystrokes. For tables, it is possible for the title, headings, and data to be read by the NVDA screen reader, but we had to manually tag each table by hand, which is an unreasonable expectation for a lot of applications. An example of manual tagging of a table is shown in Figures 3.14 and 3.15.
Figure 3.14: An example of PREP’s tagging before manual alterations
Our next task was to restructure the system’s code. Our client specified exactly how he wanted the directories sorted, so we followed his idea as closely as possible. The layout of his desired directory system is shown in Figure 3.16.

Figure 3.15: An example of PREP’s tagging after manual alterations
- **Config**
  - config.yaml - contains paths to model files, path to database (directory) containing all pdfs

- **layout_models**
  - detectron2
  - yolov7
  - yolov8 (to be added later)
  - model_utils.py (to run inference on each of the models above)

- **xml_parser**
  - xml_utils.py

- **flask_app**
  - flask_main.py
  - template
    - index.html
    - loading.html
    - upload.html
    - (more to be added later)

- **data/**
  - tmp/
    - pdfs/ – uploaded pdfs
    - page_images/ – page images
    - detected_images/ – all the image based elements like figures, tables, equations

- **database/**
  - <file_id>/
    - detected_images/ – all the image based elements like figures, tables, equations
    - <file_id>.xml - generated xml

- **api**
  - TBD

Figure 3.16: The client’s desired system architecture
Chapter 4

Implementation

In this section we will discuss the implementation process each sub-team has gone through.

4.1 Algorithms/Back-End

For the algorithms team there are two major components that we were able to implement: paragraph splitting, and adding the XML and HTML attributes (for the front-end sub-team to use for their highlighting tool).

4.1.1 Paragraph Splitting

We have two current algorithm designs that we worked with to improve the bounding boxes that are placed around both the paragraphs and the references. The first algorithm we tested takes advantage of the OpenCV [2] (CV2) library to dilate and apply contours to the text in the original paragraph/reference bounding boxes, in order to determine where a paragraph/reference starts and ends, through image manipulation. This approach was scrapped because of its inflexibility when handling documents with different structures, such as bullet points and inconsistent line spacing. A solution to paragraph splitting that would account for these using CV2 would require many conditionals, e.g., one for each possible format.

The second and more updated algorithm that we worked with takes advantage of the PDFminer [11] library where all the text of a given page, even including other non-paragraph/non-reference objects, is taken out and given their individual bounding boxes. We compared these new bounding boxes created via PDFminer to the bounding boxes returned from the object detection model to determine whether a PDFminer bbox is a paragraph/reference.

Figures 4.1 and 4.2 represent our previous logic and some of its errors. While trying the CV2 approach, we ran into issues about line spacing inside of paragraphs and realized that sentences were being treated as paragraphs. This didn’t seem to cause any issues when we merge the corresponding sentence boxes via X coordinate position, but it did raise issues when it came to bulleted lists as we can see in Figure 4.2. This was an issue we are trying to solve using our second approach.
Limitations and Delimitations

This study was restricted to African American males in their senior year of study at a PWI. The African American males participating in this study were able to overcome barriers and successfully complete a program of study while attending a PWI. The role social integration plays in the persistence decisions of African American males found in this study may not apply to women or other types of institutions or students of other ethnic backgrounds.

A delimitation of this study was the data being confined to the experiences of African American male students enrolled at one PWI in the 2009-2010 academic year. The experiences, perceptions, and programs that pertain to the undergraduate experience can vary dramatically among gender, ethnic groups, and institutions. This study cannot account for the reactions and responses of all groups about the factors that promote college persistence leading to graduation.

Figure 4.1: Algorithm using CV2 on double spaced paragraphs
Our second approach has had better success at determining paragraphs. As seen in Figure 4.3, the black box surrounding the paragraph text is generated without the requirement for any merging of bounding boxes. Though this new method has proven to be more effective, we still saw problems as Figure 4.4 shows, where certain symbols interfered with the bounding box logic, creating small boxes that don’t cover the entire paragraph.
3.2.2 Events Extractions from Space and Time

Beyond the extraction of events purely from text, there have also been several efforts to incorporate temporal and geospatial information. Sakaki et al. (2010) analyzed the statistical correlations between earthquake events in Japan and Twitter messages that were sent during the disaster time frame. An abrupt change of volume of tweets in a specific geo region indicated a potential disaster in that area. Hong et al. (2012) constructed a probabilistic graphical model that contains both a geographical component and a topical component to discover latent regions from Twitter data. Their efforts, however, are not strictly focused on event detection, as they do not consider the temporal domain. In contrast, Ritter et al. (2012) and Panisson et al. (2014) extract events into a hierarchy of types, in part utilizing the temporal information both in the text and the timestamp of the tweet itself. However, their work does not consider the spatial information explicit in geospatially tagged tweets.

3.3.4 Spatial and Temporal Boundaries

To generate the location index (i.e. \(\hat{l}\)) and time index (i.e. \(\hat{t}\)), we need to define two transformation functions that map from a real vector space to an integer space. To do so, we first divide the geographical and temporal space into a lattice within a pre-determined boundary. For geospace, a preset boundary \(B_L^L = (x_{\text{low}}, x_{\text{hi}}, y_{\text{low}}, y_{\text{hi}})\) is determined based on the data. The geon area is then divided evenly by the number of locations \(L\) to form a \(\sqrt{L} \times \sqrt{L}\) square lattice. Each cell in the lattice has a unit length of \(U_L^L = (x, y)\), with \(U_x^L = (B_{x_{\text{hi}}} - B_{x_{\text{low}}})/\sqrt{L}\) and \(U_y^L = (B_{y_{\text{hi}}} - B_{y_{\text{low}}})/\sqrt{L}\) respectively. The transformation function for location data \(f(l)\) is then defined in Equation (3.10)

\[
f(l) = [(l_x - B_{x_{\text{low}}})/U_{x}^L] \times \sqrt{L} + [(l_y - B_{y_{\text{low}}})/U_{y}^L]
\]

In the latter weeks of the project, we realized that the PDFminer algorithm approach is the best that we can do within the semester, but the logic required serious understanding and time in order to get results, so we focused purely on getting the paragraph splitting logic to work while putting the references splitting logic second.

The second approach’s algorithm had been improved in the latter weeks that we worked on the project. As seen with this updated Figure 4.4 image, Figure 4.5, the small bounding box surrounding only a bit of the text in the paragraph is now correctly bounding the entire paragraph. This has been done by removing any faulty bounding boxes that the PDFminer algorithm creates. Though the accuracy of our algorithm at correctly surrounding each paragraph increased, there have been consistent issues with any ETDs that contain
mathematical symbols inside of their paragraphs. In those cases, there are problems with the PDFminer algorithm, as shown in Figure 4.6, creating inconsistencies with our final paragraph bounding boxes that we were unable to resolve. Though this is an issue we faced until the end of our work, our algorithm seems to work with all other ETDs successfully. In the future we would hope that more testing will be conducted with other ETDs to make sure that the approach is reliable.

3.3.4 Spatial and Temporal Boundaries

To generate the location index (i.e. \( \hat{l} \)) and time index (i.e. \( \hat{t} \)), we need to define two transformation functions that map from a real vector space to an integer space. To do so, we first divide the geographical and temporal space into a lattice within a pre-determined boundary. For geospace, a preset boundary \( B^L = (x_{low}, x_{hi}, y_{low}, y_{hi}) \) is determined based on the data. The geonumber is then divided evenly by the number of locations \( L \) to form a \( \sqrt{L} \times \sqrt{L} \) square lattice. Each cell in the lattice has a unit length of \( U^L = (x, y) \), with \( U_x = (B_{xhi} - B_{xlow})/\sqrt{L} \) and \( U_y = (B_{yhi} - B_{ylow})/\sqrt{L} \) respectively. The transformation function for location data \( f(l) \) is then defined in Equation (3.10)

\[
f(l) = [(l_x - B_{xlow}^L)/U_x^L] \times \sqrt{L} + [(l_y - B_{ylow}^L)/U_y^L] \tag{3.10}
\]

Figure 4.5: Successful implementation of PDFminer on paragraph containing symbols

\[
g(\phi_{t,k}) = \prod_{d=1}^{D_t} P(w_{t,d}|\phi_{t,k}, s_{t,d} = k) = \prod_{i} \left( \sum_{j} e^{\phi_{t,k,i}} \right)^{\nu_{t,k,i}^\psi} \tag{4.11}
\]

By utilizing the notations defined above, the integration can be expressed in Equation (4.12).

Here we use \( \tau_k \) to denote the first time step when cluster \( k \) occurs. We also define \( \psi_{t,k}^\phi = 0 \) when \( t = \tau_k \) and \( \psi_{t,k}^\phi = \pi_{t-1,k} \) if \( t > \tau_k \). Similar definition can be applied to \( \psi_{t,k}^\psi \).

Figure 4.6: Latest PDFminer implementation still contains issues with paragraphs containing symbols

Another issue we faced until the end with our current implementation is the speed of our algorithm. Because of all the checks that are needed to make sure that we only include the PDFminer bounding boxes that correctly surround each paragraph, many comparisons between each bounding box generated are required. As seen in Figure 4.7, each paragraph bounding box generated by PDFminer is shown in blue, each object detected bounding box is shown in green, and each single line bounding box generated by PDFminer is shown in red. Multiple comparisons are made between each paragraph bounding box to each other in order to determine whether one is inside another. If one is inside another, then it is not considered, as it’s likely a small bounding box that covers only a single word or two.
Comparisons between the paragraph bounding box and object detected bounding box also happen in order to only consider bounding boxes corresponding to paragraph bounding boxes. Finally, single line bounding boxes are also compared to object detected bounding boxes to make sure that we don’t include any sub-headers or extraneous text that shouldn’t be included with the paragraphs.

3.3.4 Spatial and Temporal Boundaries

To generate the location index (i.e. \( I \)) and time index (i.e. \( T \)), we need to define two transformation functions that map from a real vector space to an integer space. To do so, we first divide the geographical and temporal space into a lattice within a pre-determined boundary. For geospace, a preset boundary \( B^L = (x_{low}, y_{low}, y_{high}, x_{high}) \) is determined based on the data. The geoarea is then divided evenly by the number of locations \( L \) to form a \( \sqrt{L} \times \sqrt{L} \) square lattice. Each cell in the lattice has a unit length of \( U^L(x, y) \), with \( U^L = (B^L - B^L_{low})/\sqrt{L} \) and \( U^L = (B^L - B^L_{low})/\sqrt{L} \) respectively. The transformation function for location data \( f(t) \) is then defined in Equation 3.10.

\[
f(l) = \left[ (l_x - B^L_{low})/U^L_x \right] \sqrt{L} + \left[ (l_y - B^L_{low})/U^L_y \right]
\]

Equation 3.10

Similar to the way that \( t \) is mapped to \( l \), a function that maps \( t \) into an index space \( I \) is also defined in Equation 3.11. Here we treat \( t \) as a real valued scalar bounded in range from \( B^T_{low} \) to \( B^T_{high} \). A unit length \( U^T \) is also calculated to be the unit length of each time cell in the lattice, which is \( (B^T_{high} - B^T_{low})/T \).

\[
g(t) = \left[ (t - B^T_{low})/U^T \right]
\]

Equation 3.11

In our model we treat the timestamp of a document as a real valued variable by dividing the UNIX time by the number of seconds in a month. By doing this we converted the information so that tweets are represented by a real valued variable that defines the month and year in which they occur. This meets the requirement of the Gaussian distribution in which we used to model the temporal span of a particular event.

Figure 4.7: Image containing all bounding boxes required for implementation

With all these checks and comparisons, our algorithms spend a lot of time choosing the boxes that we consider before even applying the logic, which has considerably slowed down our Flask application. A possible fix to this issue was in the making near the end of the semester by requiring only one comparison of the bounding box returned by PDFminer to the last successful addition to our paragraphs array, though this fix was not implemented so the speed issue is still yet to be addressed.
4.1.2 XML to HTML Attribute Generation

The other deliverable for the algorithm sub-team was to provide values associated with the object detection to the XML and HTML files. These values include the bounding box, confidence, and page number associated with each paragraph, figure, or heading. The previous implementation featured only the bounding boxes located within XML tags. Because the UI sub-team’s implementation of a PDF viewer is based on HTML and uses the object detection’s bounding boxes, the tags were converted into attributes within HTML. This modification to XML and HTML generation also included adding the above-mentioned values as tags in each.

For the XML generation, we pass in the four main items:

- Bounding box coordinates
- Prediction confidence percentage
- Page number
- Value (i.e., figure caption or paragraph text)

When implementing this, we used the xml.etree.ElementTree Python library to handle XML generation and parsing. Each node on the element tree consists of a tag, text, and attributes. The tag is the internal name of the object, such as ‘title’, ‘section’, or ‘paragraph’. The text is the part of the element that is displayed, and is retrieved from previous sections. When XML documents are displayed, this is what is shown. Attributes exist as a dictionary, so each piece of relevant data has their own name and value. When putting the necessary data into XML and HTML, these elements are generated as strings.

An example of an XML tag before adding the tags is in Figure 4.8.

![Figure 4.8: Image Containing all bounding boxes required for implementation](image)

This is with the XML attributes added in Figure 4.9.

![Figure 4.9: Image Containing all bounding boxes required for implementation](image)
4.1.3 Text Cleaning

Another goal is to fix issues with hyphenated text which was extracted from paragraphs. Often what would happen is that each line detected from PDFminer has a chance to be split up by a hyphen, which would denote the end of a line. We want to combine all the text together such that we do not have paragraphs split up incorrectly. So we implemented de-hyphenation by using the Python library, dehyphen [5]. Some known issues with de-hyphenation is that it takes an extremely long time to process the text, due to iterating through every word in the text and combining them into sentences. After that it will combine all the sentences into one text box. Sometimes the de-hyphen logic will come into contact with bad Unicode/ASCII characters which it cannot read, and that will slow down the process. This is compounded with the paragraph splitting logic which was discussed in Section 4.1.1. The speed of the program is severely hindered when trying to add both types of logic into the main system. Without de-hypenation it took 1 minute from inputting a PDF to generating the HTML page. With de-hypenation it took 5 minutes. Finally, with both de-hypenation and paragraph splitting it can take anywhere from 15-30+ minutes to generate the HTML.

4.1.4 Addressing Case Study Common Issues

In Appendix B.1, there were 5 common issues which were found by the accessibility team. In this section we will address some of these issues:

- **Issue**: Table of contents and list of figures are not tagged.
  **Solution**: Because the front-end has implemented a navigational sidebar, the client and front-end team team has found it sufficient to not tag the table of contents and list of figures in the ETD viewer.

- **Issue**: Paragraph splitting issues: Paragraphs are not separated from each other; they only separate when another element (e.g., header, figure, etc.) or a page break separates them.
  **Solution**: The version which was given to the Accessibility team intentionally left this feature out due to time constraints and approval of our client.

- **Issue**: Only first page of references is identified and tagged.
  **Solution**: These have been fixed so that all references are identified and tagged.

- **Issue**: All figures are a different color in PDF versus application.
  **Solution**: Saving images with OpenCV will switch the color scheme from RGB to BGR, so we had to convert the image back to RGB before saving the image to solve this issue.
• **Issue:** Clicking on figures and equation images will not highlight the image.

**Solution:** We fixed the bug where the attributes were not being properly added to these image tags.

After the case study had been conducted, the algorithms team has also implemented features which allow the user to toggle on and off between these features:

- De-hyphenation for paragraphs.
- De-hyphenation for references.
- De-hyphenation for all text.
- Splitting for paragraphs.
- Splitting for references.

With these toggles, the user is able to balance between speed with no splitting or de-hyphenation, or accuracy by using splitting or de-hyphenation. Alternatively, they may choose to benefit from the advantages of both options by employing a combination of the two.

### 4.2 UI/Front-End

The UI was initially implemented using traditional front-end means. The website layout was written in HTML, the style was written in CSS, and the dynamic components were written in JavaScript. We included some CSS from Bootstrap for the more complex components, such as the nested table of contents. The CSS and JS were written to be applied as a general solution to an automatically generated HTML document created by parsing an XML file. This XML file was generated by the Algorithms group parsing the PDF of the ETD. Later on, the client asked us to migrate the existing project into the React framework to facilitate more advanced front-end methods and in anticipation of future work.

The React framework provided our group with many options for advanced front-end development. The PDF viewer is specifically designed for the React framework and would not have been feasible to implement strictly with HTML, CSS, and JavaScript otherwise.

### 4.3 Accessibility

Our only task that required implementation was the restructuring of the system’s code. We were able to structure the code as our client wanted, by first copying the existing code into
a new sub-folder so as to not disrupt the current working code, then rearranging the code into the proper folders as directed. From there, we went through each file in the code and put each hard-coded reference to a path or directory into a YAML file, “config.yaml”. This allowed us to list all necessary folders, directories, and files in one document rather than directly editing the source code every time we moved a file. This will also help any users of the system in the future, since if they download the code, they can easily change its home directory in the config file rather than going through the full code.

One issue we ran into while restructuring the code was that images weren’t loading on the website. Through thorough testing and research, we discovered that this was due to a special requirement of Flask applications: all media that will be displayed to a website must be contained in a static folder in the same directory as the executable. However, our client specifically wanted all images related to a PDF to be stored in “data/database/<file_id>/detected_images/” to allow easy access to these images in the future. To solve this issue, while the code is running, we copy the “detected_images” folder for the given PDF into “flask_app/static/”, allowing the code to pull the images from there and display them to the web page. In order to keep the directories clean, we then remove the copy of “detected_images” from the static folder once the user has finished loading the web page. The final structure of the code is shown in Figure 4.10.
Figure 4.10: The final restructured system architecture

- api
  - TBD
- Config
  - config.yaml - contains paths to model files, path to database (directory) containing all pdfs
- data/
  - database/
    - <file_id>/
      - detected_images/ - all the image based elements like figures, tables, equations
      - <file_id>.xml - generated xml
  - tmp/
    - detected_images/ - all the image based elements like figures, tables, equations
    - page_images/ - page images
    - pdfs/ - uploaded pdfs
- flask_app
  - etd_parser.py
  - flask_main.py
  - static
    - scripts
    - styles
  - template
    - index.html
    - loading.html
    - upload.html
- layout_models
  - detectron2
  - yolov7
  - model_utils.py (to run inference on each of the models above)
- xml_parser
  - xml_utils.py
  - pdf_utils.py
Chapter 5

Testing/Evaluation/Assessment

5.1 Algorithms/Back-End

As far as testing goes, the algorithm side of the project is the one that required the largest amount of testing over a long period of time. As we continued to try and create an algorithm to split these paragraph and reference bounding boxes, special circumstances appeared with each new ETD that is added to the library. There is no standard convention which all ETDs use. Because of this, each algorithm required rigorous testing even when it seems to work completely for a single document.

Overall, our testing had proven to be effective as we can confidently identify simple bounding boxes that include multiple paragraphs without any special characters like bullet points. Because of the number of unique situations we came across when trying to find a solution to this issue, we’ve had to reassess the time it’s going to take to provide an efficient and totally accurate algorithm.

5.1.1 Future Work

Although we made progress on paragraph splitting, we have not been able to replicate this for reference splitting as we intended. Due to time constraints and unforeseen errors with paragraph splitting, we had to put reference splitting and breaking the reference down into its components for future developers of this application.

Through further invested time within our algorithm, it’s been realized that perfection is not something that can be achieved, at least with the approach that we continued with. As stated in the Implementation section, ETDs with mathematical symbols in their paragraphs created issues for the PDFminer algorithm that can’t be solved, or require so much processing that the Flask application took too much time just to render a single ETD document. We believe that our current PDFminer algorithm implementation has reached as close to perfection for paragraph splitting as we could truly reach with using PDFminer, and the only way to get 100% accurate results is to instead focus on improving the object detection model so that one day a post-processing algorithm, for splitting the paragraphs / references that the object detection model generates, will not be required.
Some other features identified that could be implemented in the future for people working on the algorithm are:

- Adding Optical Character Recognition (OCR) functionality for PDF documents that are scanned.

- Adding rules which would show all the objects detected in a document sequentially rather than image based objects and footnotes being at the bottom of each section in the ETD viewer.

- Develop an API with tools which takes advantage of the data directory by allowing retrieval of the generated HTML or XML based off of ETD_ID.

- Speed up the paragraph splitting logic or de-hyphenation logic discussed in the 4.1.1 and 4.1.3 subsections.

## 5.2 UI/Front-End

To see whether or not the following tests work, we did this mainly by running the web application and experimenting with the functionalities it has. One example is uploading multiple sample PDFs to see if the upload button still works after refactoring HTML and CSS. When we wanted to know if the article’s width changes accordingly when we hide the table of contents or resize the PDF viewer, we tried different combinations to ensure it works for every case. To confirm that the table of contents shows all of the sections and subsections, and goes to the correct part in the article when the user clicks, we went over the code and tested with several articles.

- The users are able to quickly understand the steps to generate a formatted article.

- The UI components are designed in the way that is straightforward for users to interact with.

- The users can hide and unhide the elements they want/do not want to see.

- The users are given the ability to view original document for reference in the web app and resize when they want to.

- The users can easily access the sections they are most interested in, with a mouse click.
5.2.1 Future Work

Future work for this case study would be to implement visual accessibility features. Examples given to the group include:

- Dark/Light Mode viewing
- Dyslexia-friendly font option
- Option to switch PDF viewer to show PDF with bounding boxes
- Ability to add new labeled bounding boxes in the PDF viewer

5.3 Accessibility

For our studies of how our Flask application compares to PREP tagged documents, our testing is thoroughly described in Section 3.3 of this document.

Testing for restructuring of the system’s code was completed throughout the implementation process. We completed this testing by running the code and looking through the error logs that were produced. We also tested the restructured code on multiple documents to ensure that the code was working properly and previous issues (such as failure to load images) had been fixed.

For our final task, we were to test the Flask application on 20-30 different theses/dissertations in order to study the shortcomings of our program. These academic papers came from five separate institutions: Ohio State University, UTA, Georgia Tech, Texas Tech, and Virginia Tech. These papers were selected from a dataset of academic papers provided to us by Bipasha Banerjee. We found many issues with the program, some of which existed across multiple documents. These common issues include the following:

- Table of contents and list of figures are not tagged.
- Only first page of references is identified and tagged.
- All figures are a different color in PDF versus application.
- Repeated text: same text appears in multiple different elements.
- Paragraph splitting issues: Paragraphs are not separated from each other; they only separate when another element (e.g., header, figure, etc.) or a page break separates them. (Note: This is because the version used to conduct the case studies was without the paragraph splitting due to time constraints.)
• Formatting errors regarding inset text: Lists, bullets, and other inset sections of text are included in paragraph, messing up the formatting.

• Issues with detection of footnotes

The full details of this study can be found in Appendix B.

5.3.1 Future Work

For future teams aiming to study or improve the accessibility of the application, we would first recommend fixing the errors listed above. Any of the errors listed in the full case study found in Appendix B would also be useful to improve. Finally, a group looking to improve accessibility when it comes to screen readers could implement tagging of tables and equations, that would lead to reading the content of said tables and equations, rather than only reading the caption associated with the element.
Chapter 6

User Manual

After launching this web application, the user is able to select one of the machine learning algorithms from Detectron2 [13] and YOLOv7 [12] to generate a more readable and interactive version of the original PDF version of an academic paper. Then, the user will be redirected to the upload page, where the user can upload a PDF file using the browse button or move a PDF file into the drag-and-drop section. The PDF file’s name will appear on the page. The user can click on the upload button to submit it to the server after confirmation or use another PDF instead by following the step given above.

A new page will show up to display to the user the real-time update on how much of the uploaded PDF has been processed, indicated in percentages. After that finishes, the generated text with images will show up in the middle of the page in the same structure as it is in the original document. The PDF viewer on the right can be used for cross reference. The user can drag his or her cursor horizontally to resize it. The user can access the PDF tools from the top bar in the PDF viewer. Also, when a user clicks on an HTML element in the article, the corresponding part in the PDF will be highlighted for cross-referencing.

The accordion table of contents on the left allows the user to visualize what each section is about. The user can click on each section and the document will automatically jump to that section. Besides, when the user clicks on the button in the table of contents, the subsections will be displayed. To hide or show the table of contents, click the hamburger icon on the top left. Moreover, to download the XML or PDF, the user can use the two buttons on the top right.
Figure 6.1: The final implementation of the UI
Chapter 7

Developer’s Manual

This web application is built with React front-end and Flask back-end, which responds to HTTP requests from React. It was developed in a Linux system in a virtual package environment and requires Flask installed to run the system. To set it up for development, install Flask with “pip install Flask.” To run the web app, please refer to Figure 4.10 for the folder structure. First, open two terminal windows and navigate to the “flask_app” and the “front-end” directories, respectively. In “flask_app”, run “export FLASK_APP=flask_main.py” and “flask run –host=0.0.0.0 -p <port>”. Note that the port number here should be consistent with the port number in the React app so that the front-end and back-end can communicate with each other. Then, switch to the terminal window with the front-end. There, enter “npm start”. The React app does not need to run at a particular port.

When developing on a remote server, if working on the editor that supports auto port forwarding, such as VS Code[7], the developer can click on the link from the front-end terminal window to open the website. Otherwise, the developer needs to open a terminal window and type in “ssh -L port:localhost:<port> username@host” to forward the port manually and enter “localhost:<port>” in a browser to view the web app.

In addition, different types of source files are organized in different directories with the folder names indicating the purposes of those files. When adding different types of files, they should be placed in the designated folders. The back-end works as an API that handles the PDF file process and generates information about the parsed PDF for the front-end to display. The front-end communicates with the back-end to let it know which algorithm is selected, to send the original PDF to the back-end, etc. When the back-end finishes processing the input data, the result will be displayed to the users by the front-end.
Chapter 8

Lessons Learned

8.1 Algorithms/Back-End

While we were able to complete most of the deliverables, there were many issues faced with getting to that point. One major problem that the sub-team faced when trying to implement paragraph splitting is that there was a lot of trial and error. Unlike most projects in the class, this work is more aligned with research and the re-iteration process that comes with research, i.e., coming up with a solution, attempting that solution, assessing the solution, and then finding a new one if that does not work. Many times we were stuck trying to find solutions that were not ideal, or found a major flaw in our implementation or the tools given.

While testing each of our solutions for paragraph splitting, we mostly tested on 1-2 documents. Once we started sampling more documents of different universities and disciplines, we noticed our solutions would fail more often than not. This is something that we did not learn until closer to the end of the semester. This is something that was slowing down our process. In hindsight, we needed to have a good set of ETDs which vary in university and discipline.

Another issue that occurred in our sub-team is the lack of pair-programming. What would occur is that each member has a specific task assigned to them, and implements that feature. The problem with this is that each of the members might not understand something completely, and implement the feature without fully understanding the scope or logic. When we would do pair-programming, the misconceptions are fixed, and having others bounce ideas helps with implementing a solution which everyone understands.

8.2 UI/front-end

Our team learned the importance of testing with multiple documents. Throughout our development process, we would consistently only test with one document, developing our solutions to meet the needs of that document. As we tested our solution with more documents, we noticed unexpected behavior due to different formats of the ETDs. Because of this realization, we had to redo a lot of our code to create a more general solution.

We also faced difficulty when trying to migrate the existing front-end to the React framework.
The Flask app was initially built with HTML templates of each page, all existing within the Flask app directory. When migrating to React, we had to convert the web pages to the React format, along with all dynamic functionality. The most difficult part of this was figuring out how to have React communicate with Flask. The Flask app originally sent HTTP requests to itself to allow navigation between pages. When switching to React, we had to learn how to send HTTP requests from the front-end to the back-end. We had a hard time finding adequate resources to accomplish this task on our own, so we had help from another CS4624 group that was also using a React/Flask tech stack. Our major lesson learned was to reach out to people with experience instead of trying to go solo on our own research.

8.3 Accessibility

Our main lesson learned was how to communicate better amongst our team and with our client. Because one of our tasks was to restructure the code for the program, we needed to coordinate extremely well with the other subgroups to ensure that we did not accidentally overwrite or delete any files that were currently being worked on. One of our other tasks, conducting a case study of the program’s functioning and any errors with it, also required good communication skills, as we needed confirmation that the other teams were finished working on their final tasks, and that our client wanted us to proceed with testing.
Chapter 9

Acknowledgements

We would like to thank our client, Aman Ahuja, for guiding us throughout the entire semester. We would also like to thank Bipasha Banerjee for providing the accessibility subgroup with a dataset of ETDs for their case study.
Bibliography


Appendices
Appendix A

Methodology

Figures A.1 and A.2 represent our methodology for this project and the possible actions that we are trying to allow for two kinds of users: your average student or professor, and a student or professor with some kind of impairment. Figure A.1 shows the average workflow or actions that a non-impaired user might take with our application, while Figure A.2 shows the average workflow of impaired users. These are very similar, but the differences are shown in which services are applied for each workflow, illustrated in Table A.1. This table specifies the services that we are applying to our application, what their input and output data look like, and the additional libraries, functions, and environments that were used in each.

Figure A.1: Workflow of an unimpaired user
Figure A.2: Workflow of an impaired user

Table A.1: Table indicating each service given and its attributes

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service Name</th>
<th>Input File Name(s)</th>
<th>Input File IDs</th>
<th>Output File Name</th>
<th>Output File ID</th>
<th>Libraries; Functions; Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>Access figures / sub sections in sidebar</td>
<td>HTML page</td>
<td>File1</td>
<td></td>
<td></td>
<td>HTML</td>
</tr>
<tr>
<td>1B</td>
<td>Download document</td>
<td>HTML page</td>
<td>File1</td>
<td>{ETDName}.pdf OR {ETDName}.xml</td>
<td>File1</td>
<td>Oregon(), Flask send_file()</td>
</tr>
<tr>
<td>1C</td>
<td>Access alt-text through mouse hover</td>
<td>HTML page</td>
<td>File1</td>
<td></td>
<td></td>
<td>HTML environment, XMLtoHTML()</td>
</tr>
<tr>
<td>1D</td>
<td>Filter objects by type</td>
<td>HTML page</td>
<td>File1</td>
<td></td>
<td></td>
<td>HTML environment, CSS, JS</td>
</tr>
<tr>
<td>2A</td>
<td>Use focus option</td>
<td>HTML page</td>
<td>File1</td>
<td></td>
<td></td>
<td>HTML environment</td>
</tr>
<tr>
<td>2B</td>
<td>Use screen reader on document</td>
<td>HTML page</td>
<td>File1</td>
<td></td>
<td></td>
<td>JAWS Screenreader</td>
</tr>
</tbody>
</table>
Appendix B

Case Study

B.1 Common issues

- Table of contents and list of figures are not tagged.

- Only first page of references is identified and tagged.

- All figures display as a different color in application versus PDF.

- Repeated text: same text appears in multiple different elements

- Paragraph splitting issues: Paragraphs are not separated from each other; they only separate when another element (e.g., header, figure, etc.) or a page break separates them. (Note: This is because the version used to conduct the case studies was without the paragraph splitting due to time constraints.)
1.2. Background: Research Problem

In the past decade, healthcare systems in the United States have been increasingly focused on operational efficiency to deliver healthcare with increased speed and improved safety and quality. More recently, with the development of the 1995 Consumer Assessment of Healthcare Providers and Systems program (CAHPS), the focus has shifted to the patients’ experience of care delivery (Goldstein et al., 2005b). The CAHPS assessment tool was developed to glean an understanding of what occurred at critical points in the patient care process. The focus of CAHPS is not on amenities or the satisfaction of the patient. The CAHPS tool system was used to create a hospital-based survey known as Hospital Consumer Assessment of Healthcare Providers and Systems (HCAHPS). This survey has been the foundation of what the Centers for Medicare and Medicaid Services (CMS.gov) use as the universal evaluation tool for the United States.

This program and tool provided a way to evaluate healthcare quality equally, at every hospital in the nation, for the purposes of improving the quality of care. It also has provided visibility into the patient experience and empowered patients to review the performance of their hospital through the Hospital Compare website. With the adoption of this program, physicians

Figure B.1: This is counted as one paragraph when it should be two.

- Formatting errors regarding inset text: Lists, bullets, and other inset sections of text are included in a paragraph, messing up formatting
The research model (see Figure 1.3) identifies the possible moderators that could influence the relationship of the two variables such as patient characteristics and organizational culture. This study will not measure the staff’s level of engagement, or other cultural association, but will control them through careful selection of the study site. The isolation of culture was accomplished by the following known organizational culture components and their relationship to building environment:

1. The building that the staff work in is a organization of its own. The building is managed by one hospital system.
2. The individual floor or specialty unit in that building has a team of people working together to serve the same patient population, defined by the admitted diagnosis defined as the Disease Related Group (DRG). The care delivered to these patients are similar and require staff to have similar skills.
3. The hospital is a 24 hour operating system which requires multiple shifts of staff to serve the population. The specific staff on the shift create their own culture.

Figure B.2: PDF representation of list (top) vs. application’s representation (bottom)

- Issues with detection of footnotes
B.2 Ohio

B.2.1 108017: Investigating Dimensions of Psychopathy in an Adjudicated Adolescent Sample: The Role of Race, Sex and Disruptive Family Processes

- Paragraph splitting issues
- Repeated text

psychopathy provided the best fit for Caucasian males, consistent with the results of Cooke and Michie (2001). The model developed with Caucasian males also fit well for samples of Caucasian females, African-American males, and African-American females. The measurement model predicted a significant amount of variance in criminal behavior as well as a number of variables related to externalizing and internalizing symptoms. The Impulsivity/Conduct Problems factor was strongly and consistently related to all of these outcome variables, suggesting it represents a risk factor for both externalizing and internalizing psychopathology. The Callous-Unemotional Traits factor was also related to antisocial behavior, albeit less so than Impulsivity/Conduct Problems. Narcissism was positively related to violence. A few noteworthy race and sex differences emerged. First, the model predicted outcome variables as well or better for females as it did for males. Second, the model predicted serious crime less well for African-Americans than for Caucasians. Overall, results suggest psychopathy is a valid and useful construct for understanding youth antisocial behavior across sex and race. Including additional items for each of the three psychopathy factors in order to ensure sufficient coverage of each dimension may enhance the GRAD as a risk and needs classification device for adolescent offenders. Future research with the GRAD would also benefit from the inclusion of well-validated measures of externalizing and internalizing psychopathology as well as collateral reporters.
offenders. Future research with the GRAD would also benefit from the inclusion of well-validated measures of externalizing and internalizing psychopathology as well as collateral reporters. Psychopathy provided the best fit for Caucasian males, consistent with the results of Cooke and Michie (2001). The model developed with Caucasian males also fit well for samples of Caucasian females, African-American males, and African-American females. The measurement model predicted a significant amount of variance in criminal behavior as well as a number of variables related to externalizing and internalizing symptoms. The Impulsivity/Conduct Problems factor was strongly and consistently related to all of these outcome variables, suggesting it represents a risk factor for both externalizing and internalizing psychopathology. The Callous-Unemotional Traits factor was also related to antisocial behavior, albeit less so than Impulsivity/Conduct Problems. Narcissism was positively related to violence. A few noteworthy race and sex differences emerged. First, the model predicted outcome variables as well or better for females as it did for males. Second, the model predicted serious crime less well for African-Americans than for Caucasians. Overall, results suggest psychopathy is a valid and useful construct for understanding youth antisocial behavior across sex and race. Including additional items for each of the three psychopathy factors in order to ensure sufficient coverage of each dimension may enhance the GRAD as a risk and needs classification device for adolescent offenders. Future research with the GRAD would also benefit from the inclusion of well-validated measures of externalizing and internalizing psychopathology as well as collateral reporters.

psychopathy provided the best fit for Caucasian males, consistent with the results of Cooke and Michie (2001). The model developed with Caucasian males also fit well for samples of Caucasian females, African-American males, and African-American females. The measurement model predicted a significant amount of variance in criminal behavior as well as a number of variables related to externalizing and internalizing symptoms. The Impulsivity/Conduct Problems factor was strongly and consistently related to all of these outcome variables, suggesting it represents a risk factor for both externalizing and internalizing psychopathology. The Callous-Unemotional Traits factor was also related to antisocial behavior, albeit less so than Impulsivity/Conduct Problems. Narcissism was positively related to violence. A few noteworthy race and sex differences emerged. First, the model predicted outcome variables as well or better for females as it did for males. Second, the model predicted serious crime less well for African-Americans than for Caucasians. Overall, results suggest psychopathy is a valid and useful construct for understanding youth antisocial behavior across sex and race. Including additional items for each of the three psychopathy factors in order to ensure sufficient coverage of each dimension may enhance the GRAD as a risk and needs classification device for adolescent offenders. Future research with the GRAD would also benefit from the inclusion of well-validated measures of externalizing and internalizing psychopathology as well as collateral reporters.

Figure B.3: The 2nd page of the abstract PDF (top) is repeated twice in the Flask application (bottom).

- Formatting errors regarding inset text.
- Figures display in different places in PDF versus application.
B.3 Georgia Tech

B.3.1 41609: Measuring the impacts of hospital nursing floor and patient room layouts on patients’ experience with care in a major teaching hospital

- The acknowledgements are incorrectly identified as the abstract, and the entire acknowledgements section is duplicated.

- List of figures are not tagged.

- Strange tags are sometimes included.

- Paragraph splitting issues

- Figures display in different places in PDF versus application.

- All figures are a different color in PDF versus application.

- Formatting errors regarding inset text.
The research model (see Figure 1.3) identifies the possible moderators that could influence the relationship of the two variables such as patient characteristics and organizational culture. This study will not measure the staff’s level of engagement, or other cultural association, but will control them through careful selection of the study site. The isolation of culture was accomplished by the following known organizational culture components and their relationship to building environment:

1. The building that the staff work in is a organization of its own. The building is managed by one hospital system.
2. The individual floor or specialty unit in that building has a team of people working together to serve the same patient population, defined by the admitted diagnosis defined as the Disease Related Group (DRG). The care delivered to these patients are similar and require staff to have similar skills.
3. The hospital is a 24 hour operating system which requires multiple shifts of staff to serve the population. The specific staff on the shift create their own culture.

variables, noted through research and evidence in the field, are defined by the author through observation on the unit. The research model (see Figure 1.3) identifies the possible moderators that could influence the relationship of the two variables such as patient characteristics and organizational culture. This study will not measure the staff’s level of engagement, or other cultural association, but will control them through careful selection of the study site. The isolation of culture was accomplished by the following known organizational culture components and their relationship to building environment: 1. The building that the staff work in is a organization of its own. The building is managed by one hospital system. 2. The individual floor or specialty unit in that building has a team of people working together to serve the same patient population, defined by the admitted diagnosis defined as the Disease Related Group (DRG). The care delivered to these patients are similar and require staff to have similar skills. 3. The hospital is a 24 hour operating system which requires multiple shifts of staff to serve the population. The specific staff on the shift create their own culture. All information gathered for this study is tied back to one hospital and one building on the campus of the Emory University Hospital (EUnH). The next level of organizational culture is the unit. The 17 units in this study are on the same campus and vary by size, shape, and specialty. These units have various spatial layouts and different organizational cultures. The preliminary field study was conducted on two units that contained the same patient population as defined by their disease related group. The disease related group is the third level of cultural influence. The patient’s diagnosis will shape the care plan for that patient and some diagnoses have more

Figure B.4: PDF representation of list (top) vs application’s representation (bottom)

- Table 3-5 caption incorrectly identified in application
Figure B.5: Labeling on PDF (top) vs. labeling on application (bottom). "Established values by questions" is identified as table label rather than “Table 3-5 ...”

- Repeated text

Figure B.6: Note the word “performance”, should not be included in paragraph element.

- Some headings are missing. E.g., 4.3.4 and 4.3.5 headings are not included.
- Spacing of references is off.

References

B.3.2 51993: DEVELOPING AND REFINING LINKS AS A REPRESENTATION OF ORGANIZATIONAL CONNECTION

- Date identified as “Institute of Technology May, 2017” instead of just “May, 2017”
- Table of contents and list of figures are not tagged.
• Math symbols in PDF text not displaying properly in application text.

![Example text](image)

Figure B.7: PDF (top) vs. application (bottom)

• Italics do not transfer from PDF to application

• Some parts of appendix marked as text, while other parts are marked as figures, despite having the same formatting.

• Some text in appendices marked as both paragraph text and figure labels

• Only first page of references is identified and tagged.
B.4 Texas Tech

B.4.1 261721: The Wide Mouth of Bone: Original Poetry

- Two abstracts identified: first is correct, second is a random paragraph pulled from the middle of the PDF.

Abstract

The Wide Mouth of Bone: Original Poetry is a collection of poems which explores the landscape of the Llano Estacado and its relationship to memory, space, and emotion. Ache, the central theme of these poems, manifests in a number of settings and images: a bird dying in a field, the inescapable loss of a loved one, and the realization of love, to name a few. As a whole, this collection follows the path of the seasons and examines how one’s experiences become a greater conversation about our place within the natural world.

Abstract

Cannot be explained in photographs: a white and black baby waking on a black and white shoulder, loud laughter dissolving behind Floydada railroad tracks, three farmhands—the middle one sober—simple shoes, a lost mandolin, muddy muslin dresses. Sleeping in an indolent railroad bunk, broken bone buttons, and sallow winter wheat. A death cannot be explained in photographs, integrated on mantles and shelves, anesthetized, confusing the faces that survived, the boxes that lied. Cannot be explained in photographs: a white and black baby waking on a black and white shoulder, loud laughter dissolving behind Floydada railroad tracks, three farmhands—the middle one sober—simple shoes, a lost mandolin, muddy muslin dresses. Sleeping in an indolent railroad bunk, broken bone buttons, and sallow winter wheat. A death cannot be explained in photographs, integrated on mantles and shelves, anesthetized, confusing the faces that survived, the boxes that lied.

- Paragraph splitting issues

- Formatting errors regarding inset text.

- Some poems are identified as figures rather than text. (Chapter 2, chapter 4). One such case identifies all but the last 4 lines of a poem as a figure, while the last 4 lines are text as a paragraph.

- Some poem titles are identified as headings, while others are identified as part of a paragraph, while others still are missed entirely.

- An entire page is not tagged: Chapter 4’s poem “Hips” is entirely omitted from the application’s output.

- Roman numerals on pages 36-40 are not detected.

- Header is included in a paragraph.
Issues with references similar to previous documents.
B.5 Virginia Tech

B.5.1 494947: In-Between: Architectural Drawing and Imaginative Knowledge

- Date, which is in different font from rest of PDF text, is not detected.
- Table of contents and list of figures are not tagged.
- “IMAGINATIVE DRAWING AND THE IN-BETWEEN” heading not tagged; lines from paragraph text marked as heading.
- Footnotes throughout PDF not detected or tagged.
- Repeated text
- Footnotes or footnote reference numbers often identified as paragraph text.

![Image](image_url)

Figure B.8: PDF (top) versus application (bottom). Application displays the number 113 below the text.

- Page 67 figure label identified as both figure label and paragraph. Similar issue on page 171.
• Extra text identified as part of figure label.

Figure B.9: Text should start at “Fig.2.18”

• Occasionally speakers of quotes are tagged as part of text; other times they are missed entirely.

• Page 158’s footnote is falsely identified as a header.

• W-Z of glossary tagged as paragraph text, while rest of glossary is figures

• Only first page of references is identified and tagged.