Final Report

CS 5604: Information Storage and Retrieval

Team 3: Object Detection and Topic Modelling

Amr Ahmed Aboelnaga, Anushka Sivakumar, Jayanth Narla, Pradyumna Upendra Dasu,
Ragul Seetharaman, Sahana Bhaskar, Shankar Srinidhi Srinivas

Guided by

Instructor: Dr. Edward. A. Fox

Subject Matter Expert: Chenyu Mao

January 8, 2024

Virginia Polytechnic Institute and State University
Blacksburg, Virginia, 24061 USA
Under the guidance of Dr. Edward A. Fox, the CS 5604: Information Storage and Retrieval class (Fall 2023) was tasked with developing a cutting-edge information retrieval system to facilitate Electronic Theses and Dissertations (ETDs). We used learning algorithms on a large ETD collection to classify closely related documents. The project’s overarching objective is to enhance the already available service, which enables users to upload, search, and retrieve ETDs along with their associated digital objects in a human-readable format. Our team’s specific assignment is to use object detection and topic modeling to analyze documents and thereby assist in building a system that supports searching and retrieving documents using topics and user defined digital objects, and enables experimenters to conduct further research into objects and topics. To achieve this effort we have implemented object detection on 200 segmented ETDs and topic modeling using BERTopic (BERT embeddings) and LDA (Latent Dirichlet Allocation) on nearly 334k ETDs. The object detection and topic modeling pipelines have been modified to utilize APIs (Application Programming Interfaces) for populating database tables related to ETDs. Each ETD page is converted into an image and stored in the file system, with corresponding entries in the database. Additionally, all detected objects are stored both in the database and the file system. The generated XMLs now include an object ID for each detected object, facilitating the capture of structural relationships using knowledge graphs (Team 1). Efforts have also been invested in enhancing chapter segmentation in XMLs. This involves exploring and experimenting with the LLaMA 2 model, ResNet model, and clustering approaches to accurately identify the start and end pages of chapters. The topic modeling results using BERTopic were not satisfactory leading to exploration of the LDA model. Switching to the LDA model has provided promising outputs. The topics generated using LDA were refined using various pre-processing techniques and given to team 6 to be used in the sign-up page, and to team 2 for indexing.

Keywords: ETD, electronic theses and dissertations, deep learning, object detection, topic
modeling, chapter segmentation, probabilistic models
Contents

List of Figures viii

List of Tables xi

1 Overview 1

1.1 Introduction ................................................. 1

1.2 Project Management ......................................... 2

1.3 Challenges Faced and Solutions .............................. 3

1.4 Future Work ................................................ 5

2 Literature Review 7

2.1 Object Detection ............................................. 7

2.2 Topic Modelling ............................................. 8

2.3 Tools ...................................................... 9

2.3.1 PyTorch .................................................. 9

2.3.2 Detectron2 ............................................... 10

2.3.3 YOLOv7 .................................................. 11

2.3.4 LDA ....................................................... 13

2.3.5 ProdLDA .................................................. 13

2.3.6 NeuralLDA ............................................... 14
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.7</td>
<td>CTM</td>
<td>14</td>
</tr>
<tr>
<td>2.3.8</td>
<td>BERTopic</td>
<td>15</td>
</tr>
<tr>
<td>2.3.9</td>
<td>pdf2image</td>
<td>15</td>
</tr>
<tr>
<td>2.3.10</td>
<td>Flask</td>
<td>16</td>
</tr>
<tr>
<td>2.3.11</td>
<td>OCTIS</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Requirements</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Data Requirements</td>
<td>18</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Dataset Cleaning</td>
<td>18</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Object Detection</td>
<td>18</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Topic Modelling</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Training</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>Usage</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>Processing Requirements</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>Design</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Approach</td>
<td>22</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Object Detection</td>
<td>22</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Topic Modeling</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Deliverables</td>
<td>24</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Object Detection</td>
<td>24</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Topic Modeling</td>
<td>26</td>
</tr>
</tbody>
</table>
5 Implementation

5.1 Milestones ......................................................... 28

5.2 Object Detection .................................................. 29

5.3 Heuristic Chapter Segmentation ............................... 40

5.4 Automated Identification of TOC Pages Using ResNet Model .... 45

5.4.1 Description ..................................................... 45

5.4.2 Pseudocode for ResNet Model Application ................. 46

5.4.3 Evaluation and Results ....................................... 46

5.4.4 Application on the ETD 500 Dataset ....................... 48

5.4.5 Combining ResNet Approach with Algorithm 2 for Chapter Extraction 49

5.5 Topic Modeling ..................................................... 52

6 Future Work ......................................................... 61

6.1 Object Detection .................................................. 61

6.2 Topic Modeling .................................................... 63

7 User Manual ........................................................ 64

7.1 Object Detection .................................................. 64

7.2 Topic Modelling .................................................. 65

7.2.1 Prerequisites ................................................... 65

7.2.2 Website ........................................................ 65

8 Developer Manual ................................................... 71
8.1 Object Detection .................................................. 71
8.2 TOC Segmentation .................................................. 73
8.3 Topic Modeling ..................................................... 74
  8.3.1 Prerequisites .................................................... 74

Bibliography .......................................................... 77
List of Figures

2.1 Object detection using Detectron2 [13] ....................................... 10
2.2 YOLOv7 performance compared to other object detectors [18] ............ 12
2.3 YOLOv8 performance compared to older versions [17] ...................... 13
2.4 Sample Code Snippet for Converting PDF to Image using PDF2Image library. 16

4.1 Object Detection - from ETDs to the XML/JSON ................................ 23

5.1 An example snippet of the XML schema ........................................... 31
5.2 An example snippet of the front section of the XML schema of ETD [7] ... 32
5.3 An example snippet of the body of the XML schema of ETD [7] ............. 32
5.4 An example snippet of details associated with the image-based objects in the body of the XML schema of ETD [7] ............................................. 33
5.5 An example snippet of the back section of the XML schema of ETD [7] ... 33
5.6 An example snippet of the XML schema of ETD [12] with Object ID highlighted for each object detected. ............................................. 34
5.7 Maximum resource utilization for batch size = 10 ............................... 37
5.8 Maximum resource utilization for batch size = 50 .............................. 37
5.9 Maximum resource utilization for batch size = 75 .............................. 38
5.10 Current architecture for bulk processing ......................................... 39
8.3  flask_app ............................................................... 73

8.4  xml_parser ............................................................. 73

8.5  Team 3 container on the Endeavour cluster with model path ....... 74

8.6  Team 3 container on the Endeavour cluster with root directory .... 74

8.7  Team 3 container on the Endeavour cluster for Topic Modeling. ....... 76
List of Tables

5.1 Object Detection and Chapter Segmentation Milestones ........................................ 29
5.2 GET APIs for Object Detection ................................................................................. 35
5.3 POST APIs for Object Detection ................................................................................ 36
5.4 Topic Modeling Milestones ......................................................................................... 53
5.5 POST APIs for Topic Modeling ................................................................................... 54
5.6 Summary of Topic Modeling experiments ................................................................. 58
Chapter 1

Overview

1.1 Introduction

A thesis or dissertation is a scholarly documentation of the original research conducted by students, typically as a requirement to obtain their graduate degrees. In the past, the theses and dissertations were documented and stored on paper which made them inaccessible to other scholars and researchers interested in the field. In the year 1997, Virginia Tech became the first university to mandate the recording of these precious artifacts electronically. The shift in the modality of storage has contributed greatly to the scholarly community by improving the accessibility of the information in the artifacts.

Our goal over the course of the semester was to build on top of the existing architecture for improving the efficiency of storing the Electronic Theses and Dissertations (ETDs) and make them more accessible by improving the organization of the information. To accomplish this, the CS5604 class strived to build auxiliary systems to model topics of ETDs for better searching of related work, extract and segment relevant information like metadata, figures, chapters, table of contents, etc., and also provide summarizations of chapters in support of streamlining different tasks for different personas like researchers, experimenters, and general users. There are six teams responsible for different components and services for the overall system. Team 1 is responsible for generating knowledge graphs that model the structural/relational information for the ETD based on the information extracted during object detection by Team 3. Team 2 is responsible for indexing, searching, and ranking ETDs.
Team 3 is responsible for extraction of image-based and text-based objects from ETDs, and topic modeling. Team 4 is responsible for summarizing and classifying the chapters and their contents to effectively describe them. Team 5 is responsible for maintaining the file system and database, providing APIs (Application Programming Interfaces) to perform actions on the data, and managing the overall infrastructure for the system. Team 6 is responsible for the User Interface to support all personas and define how the user interacts with the system.

In this report, we discuss the progression of our team’s work (Team 3). The primary focus was to develop the auxiliary services for the experimenter and curator personas to analyze the documents by performing object detection and topic modeling which will enable searching of ETDs through chapters or specific topics. We also discuss the architecture, and plans for implementation of the algorithms we utilized. Team 5 provided the support to store and retrieve the segmented objects from the ETDs, as well as the list of topics generated by the topic modeling pipeline, by utilizing a sequence of API calls. Figures 4.1 and 5.20 shows the overall pipeline that our team has been following.

The object identification system utilized the YOLO (YOLOv7) and Faster R-CNN (Detectron2) models to leverage their capabilities. Additionally, post-processing techniques were employed to analyze and interpret the text and image objects. The topic modeling system used methods like LDA and BERTopic to associate ETDs to different topics. The complete pipeline will be made clear in the following sections.

1.2 Project Management

Since our team has responsibilities of two distinct components, team members are often required to be involved in both. Effective task management has become essential for enhancing productivity. Our project utilizes Trello’s functionalities to improve team member collaboration and expedite operations. We have created a Trello board with labels such as
“To-Do,” “In Progress,” and “Completed” to represent the various stages of our Endeavour. Each task or deliverable is represented by a card that is moved across these categories to indicate its current status. To ensure timely completion, team members are assigned individual cards with plainly marked deadlines. Further, to track and manage changes made to the software by the team members, we are using GitLab as our version control system. We conduct weekly meetings to track project progress. During these meetings, team members provide updates on their progress, discuss future tasks, and address any challenges while planning steps for issue mitigation. We also hold discussions with the Subject Matter Experts in class to get feedback on our progression towards the goals, and insights into the existing implementation of the systems.

1.3 Challenges Faced and Solutions

During the course of this work, we encountered several challenges that required solutions. The professor, the subject matter experts, as well as the teaching assistant for the course have assisted us in laying out the process while removing barriers in each step. The following are some of the solutions for the challenges faced.

1. Setting up of the previous code base in the Endeavour cluster, especially due to version mismatch of various packages and inconsistencies between pip installed packages and conda installed packages. Here, the previous code base was successfully set up in the Endeavour cluster by cross-verifying the package versions installed in the conda environment as well as in Python by referring to error logs and online forums to debug installation errors.

2. Uncertain erasure of conda environments in the container as a result of rebooting each node to fix the failed updates to the underlying kubernetes system, and container crashes that led to problems with cluster networking. To deal with the uncertain
erasure of the conda environment, we documented the conda environment set up to ensure speedy set-up in case the environment was erased again. Additionally, on the advice of Dr. Fox and Chris Arnold, we worked on avoiding package updates to a running container on Kubernetes, and focused on installation outside of the cluster to the Docker image itself.

3. Issue with the initialization of the Faster-RCNN Detectron2 trained model. We were unable to resolve the installation issues with respect to the config file for the Detectron2 model. From the report by the Fall 2022 CS 5604 team working on object detection [8], YOLOv7 showed better performance than Detectron2 on the collection of ETDs. Hence, we decided to implement only the YOLOv7 model for the scope of this work.

4. Stalled efforts to populate the database and file system, and generate XMLs for the ETDs processed through the object detection pipeline due to issues with API implementations. Since then, we continued to work closely with Team 5 to ensure complete integration of the APIs into the object detection pipeline.

5. Stalled efforts in implementing Kafka due to the prioritization of other more important tasks and infrastructure setup. We worked on continually improving the proposed architecture for the Kafka implementation of the object detection model based on discussions with Dr. Fox, the subject matter experts, and the TA so that it could be ready to be implemented when the architecture allows for it.

6. Long running jobs like topic modeling training on the ETD dataset used a Virtual Machine (hostname: dlrl) provided by Satvik Chekuri, an SME as part of the CS 5604 course who is also a Ph.D. student in the DLRL (Digital Library Research Laboratory). This was done because of the model training stopping in between and sometimes not starting at all. Since the root cause was not known, we have decided to train the model and move all the results and scripts that can be used to the Endeavour cluster
1.4 Future Work

This project is a work in progress and has large scope for improvements in accuracy and scalability. Following are brief descriptions of some of the improvements in the scope of the object detection and topic modeling pipeline in this work. The Future Work is elaborated in Chapter 6.

1. Improve the object detection model accuracy and generalizability.

2. Experiment with newer object detection models for ETD data.

3. Explore a top-down object detection approach and its benefits in generating the XML for the ETD. A top-down object detection approach which provides a global view of an image to identify objects, can be beneficial for generating XML for an ETD. The structured data in XML format, resulting from this approach, is both machine-readable and human-readable, enhancing the usability and accessibility of the ETD.

4. Ensure accurate handling and processing of scanned PDF ETDs and born PDF ETDs.

5. Incorporate the chapter segmentation code with the object detection pipeline. An XML with well-segmented chapters will then be passed on to Team 4 for classification and summarization tasks.

6. Set up Kafka for batch processing.

7. Populate both the database and the file system with the extracted data and files generated by passing the ETD through the object detection pipeline using the integrated APIs.

8. Complete processing 500,000 ETDs through the object detection pipeline and the topic modeling pipeline.
9. For topic modeling, analyze and improve the usage of BERTopic on the cleaned ETD dataset; this is an important future goal.

10. Analyze and improve the keywords formed by the LDA model by updating custom stopwords while also trying different vectorizers other than tf-idf. Further, we want to explore different lemmatization methods.

11. Explore other topic modeling approaches such as ProdLDA and NeurLDA on the large ETD dataset.

12. While signing up, scanning through 70 generated topics is not a user friendly approach. A better approach like hierarchical topic modeling needs to be considered.

The report is organized as follows: Chapter 2 performs a literature review, highlighting the tools and methodologies used for object detection and topic modeling. Chapter 3 provides details on the project’s requirements concerning the dataset, training models, usage, and processing capabilities. The current designs for object detection and topic modeling are elaborated in Chapter 4. This chapter also includes information about the previous architecture, proposed improvements, and deliverables aimed at enhancing the model.

Chapter 5 outlines the milestones achieved during this course and delves into the experiments, including both failures and successes, as well as the implementations made in CS 5604 for object detection, chapter segmentation, and topic modeling. Our future plans for improving and expanding the current model are described in Chapter 6.

Finally, Chapter 7 and Chapter 8 detail the execution of the object detection model and topic modeling model from the perspective of users and its navigation from the perspective of developers, respectively.
Chapter 2

Literature Review

The field of information storage and retrieval has seen significant advancements. Yet, there remain gaps in its application to large Electronic Theses and Dissertations (ETDs) datasets. This literature review aims to explore current methods in object detection and topic modeling, focusing on their relevance and limitations for handling ETD collections. We conducted a methodical search of recent studies using databases like Google Scholar and the ACM Digital Library. The review is organized into sections that cover the fundamental object detection technologies, and the various topic modeling approaches. Through this examination, we seek to inform the design of an advanced ETD retrieval system in collaboration with Virginia Tech’s University Libraries.

2.1 Object Detection

We implement the object detection process with two models, namely YOLOv7 [22] and Faster R-CNN [19]. The third version of Ross Girshick et al.’s R-CNN architecture is called Faster R-CNN, and it was first released in 2015. R-CNN’s Region of Interest (RoI) Pooling was used in a second version of the algorithm known as Fast R-CNN to replace the selective search technique. This allowed for the pooling of costly calculations and significantly accelerated inference. With the use of the CNN’s feature maps, the Region Proposal Network (RPN), which was introduced by Faster R-CNN, can be trained in tandem with the CNN to produce high-quality suggestions. It should be noted that the original Faster R-CNN implementation
used a standard VGG-16 backbone architecture for the CNN; however, the framework we are using, Facebook AI Research’s Detectron2, contains object detection models built on top of Faster R-CNN with sophisticated architectures that perform incredibly well [23]. Even though Faster R-CNN was the most advanced model for many years, newer YOLO models and other models with improved speed vs. accuracy trade-offs have supplanted it. When YOLOv7 is compared to other real-time object detectors, it outperforms both transformer-based and convolution-based object detectors, with the highest accuracy of 56.8 percent average precision [22].

2.2 Topic Modelling

Natural language processing (NLP) and text mining have both made significant strides in recent years. The goal of topic modeling in ETD analysis is to identify thematic clusters of words from a large corpus of text documents that may be used to represent themes. Over the years, numerous topic modeling algorithms have been put forth. Previous research, such as with Latent Dirichlet Allocation (LDA) [6], relied on the concept of topic formation based on probability. In the field of topic modeling, recent developments include neural topic models like CTM (that allows for correlation between topics) [5], NeuralLDA (to model topics as continuous vectors), ProdLDA (using Variational Autoencoders (VAEs)) [20], and BERTopic (using transformers and c-TF-IDF to create dense clusters allowing for easily interpretable topics) [11]. The representations that are acquired through topic models can be applied to downstream tasks that depend on document representations, like document recommendation, locating documents by topic, topic by university, and examining topic fluctuation over time.
2.3 Tools

Below is a list of the tools used for our work. This is followed by a subsection for each of those tools giving more detail.

1. PyTorch

2. Nvidia GPU

3. Docker Engine

4. Detectron and YOLOv7, OCR

5. LDA, Neural LDA, ProdLDA, CTM, BERTopic

6. OCTIS

2.3.1 PyTorch

The Facebook AI Research lab (FAIR) created PyTorch, an open-source deep learning framework [16]. It is employed for numerous machine learning and artificial intelligence tasks, specifically in the area of deep learning. The core data structures of PyTorch are tensors. They are the basic units for constructing and modifying neural network models. Tensors can be used on GPUs to enable faster processing. With PyTorch, the graph is constructed dynamically as operations are carried out. This dynamic nature enables more flexible and
understandable model development, making it simpler to troubleshoot and test out various network structures.

2.3.2 Detectron2

The newest library from Facebook AI Research, Detectron2, offers cutting-edge object segmentation and detection algorithms. It is the replacement for the original Detectron framework and is well known for its computer vision applications, adaptability, modularity, and excellent performance. Detectron2 is developed on top of PyTorch and it provides a full suite of tools and pre-made components for object detection. It enables faster training of models. Another benefit of Detectron2 is that it allows models created with it to be deployed in TorchScript or Caffe2 format. Detectron2 offers pre-trained models on expansive datasets like COCO (Common Objects in Context). As a starting point for transfer learning, these pre-trained models enable you to fine-tune models on your datasets with only a minimal quantity of data. We will be using this library for carrying out object detection and classifying the pages of the ETD into multiple text and image based objects. An example of the object detection using Detectron 2 can be seen in Figure 2.1.

![Diagram of Classification, Semantic Segmentation, Object Detection, and Instance Segmentation](image)

*Figure 2.1: Object detection using Detectron2 [13]*
2.3.3 YOLOv7

YOLOv7 is one of the state of the art real time object detection models, excelling at detecting various objects in real time. As mentioned in the official paper written by Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao [22], YOLOv7 operates efficiently in a frame rate range of 5 FPS to 160 FPS and boasts the highest accuracy score of 56.8% AP among real-time object detectors with 30 FPS or higher on a GPU V100. The model significantly outperforms other object detectors in both speed and accuracy metrics, even when compared to both transformer-based and convolutional-based models like SWINL Cascade-Mask R-CNN and ConvNeXt-XL Cascade-Mask R-CNN. Notably, YOLOv7 was trained exclusively on the MS COCO dataset without leveraging any other datasets or pre-trained weights. YOLOv7 performance compared to other object detection models can be seen in Figure 2.2.

YOLOv7 employs a convolutional neural network to simultaneously generate bounding boxes and class likelihoods by analyzing the entire image in a single go. This is in contrast to traditional object detection algorithms, which focus on specific regions of the image to pinpoint objects and their associated probabilities. The YOLOv7 architecture consists of several layers: firstly, the image frames pass through a ‘backbone,’ which is essentially a deep neural network made up of numerous convolutional layers designed to extract specific features. These features then flow through the ‘neck,’ an intermediary set of layers that integrates and collates the feature maps extracted at various stages by the backbone. This is followed by the ‘head,’ where actual object detection takes place via dense prediction methods. The backbones implemented in YOLOv7 include Vision-Only Volumetric Networks (VoVNET), Cross-Stage Partial VoVNET (CSPVoVNET), Efficient LAttice Networks (ELAN), and Extended-Efficient LAttice Networks (E-LAN). As for the neck, it utilizes Feature Pyramid Networks (FPN), Receptive Field Block (RFB), and Path Aggregation Network (PAN). Finally, the detection head makes simultaneous localization and classification predictions, setting it apart from two-stage detectors like Faster R-CNN and Region-based Fully Convolutional Networks.
(RFCN), which perform sparse predictions to establish class probabilities.

Figure 2.2: YOLOv7 performance compared to other object detectors [18]

There is an unofficial new version of YOLOv7, called YOLOv8. Future work includes experimenting object detection with the YOLOv8 model due to its improved performance capabilities as seen in Figure 2.3.
2.3.4 LDA

Latent Dirichlet Allocation (LDA) [6] is a probabilistic generative model used in machine learning and natural language processing for document grouping and topic modeling. It envisions a generative process in which subjects and words are chosen at random from a set of topics to produce documents. Reverse engineering this generative process is the aim of LDA. LDA determines the subjects that produced a collection of documents and the distribution of words within each topic. Usually, probabilistic inference widths like variational inference or Gibbs sampling are used for this. The distribution of document topics and the distribution of topic words are the two basic groups of parameters in LDA [6].

2.3.5 ProdLDA

A probabilistic topic modeling approach for massive amounts of text data is called ProdLDA, [20] or Product of Latent Dirichlet Allocation. By constructing numerous LDA-based “experts,” each modeling a subset of topics, it divides the topic modeling challenge into more manageable, smaller subproblems. By merging the findings from multiple experts through a product operation, the final topic distribution is generated. With the help of this width,
scalability and parallelization are improved, making it effective for processing big-text corpora and identifying relevant subjects in vast datasets. Several natural language processing applications benefit greatly from ProdLDA’s hierarchical structure and complexity reduction.

2.3.6 NeuralLDA

Neural LDA (Latent Dirichlet Allocation) is an extension of conventional LDA. It integrates neural networks into the topic modeling procedure. The representation of documents and subjects is improved by the use of neural networks, typically deep learning models. By capturing semantic linkages in text data, this width enables more intricate and context-aware topic modeling. Compared to classic LDA, neural LDA is more adaptable because it can automatically learn a variety of themes. It is frequently utilized in contemporary text analysis jobs as it can improve performance and adaptability.

2.3.7 CTM

Correlated Topic Model (CTM) [5], a probabilistic topic modeling method, is used to capture correlations between themes in a group of texts. Although CTM assumes that documents are produced using a variety of topics, it goes a step further by enabling topics to be associated using a common topic correlation matrix. Due to the ability to model topic interdependence, CTM is appropriate for jobs in which topics are not independent. To calculate topic-word distributions and topic correlations, CTM uses probabilistic inference techniques. It has uses in a number of NLP tasks, including content recommendation, document classification, and information retrieval, where topic relationships must be captured for better modeling and comprehension of document collections.
2.3.8 BERTopic

BERTopic [11] is a topic modeling technique that uses the transformer-based language model – BERT (Bidirectional Encoder Representations from Transformers) – for topic modeling. It could overcome some of the shortcomings of traditional topic modeling methods by leveraging the power of contextual word embeddings provided by pre-trained transformer models. It has the capability to create highly context-aware word embeddings and capture the meaning of words in the context of the entire text. BERTopic applies dimensionality reduction techniques like UMAP (Uniform Manifold Approximation and Projection) to reduce the high-dimensional BERT embeddings into a lower-dimensional space. It performs clustering using methods like HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise, to find clusters of varying shapes [14], which effectively groups the documents into distinct topics.

2.3.9 pdf2image

This is a Python library that offers a quick and easy approach to turn PDF files into a series of images to perform operations like document processing, automation, content extraction, OCR (Optical Character Recognition), or image processing. This library provides an API that makes it simple to incorporate into our Python programs. The output path, other parameters, and the input PDF file can all be specified as needed like in Figure 2.4. We are incorporating this tool in the data pre-processing step where we will be extracting images from the ETDs.
2.3.10 Flask

Integrating the Flask framework with our back-end code is easy and convenient because it is a lightweight, Python-based web framework based on WSGI [10]. Because it is a micro-framework design, we may combine it with any wheels or Python libraries we require for the project development, and it is highly scalable and versatile thanks to the large range of third-party libraries. Furthermore, it would be simple to develop new online features and functionalities. In addition, front-end technologies like React [15] are required. The fundamental framework of the website, its styling, and user interaction will be managed by Team 6.

2.3.11 OCTIS

The acronym OCTIS [21] refers to optimizing and comparing topic models. It is a Python program that makes this possible and facilitates the analysis of ETD topics. LDA is now

```python
from pdf2image import convert_from_path

# Specify the PDF file you want to convert
df_file = "example.pdf"

# Convert PDF to a list of images
images = convert_from_path(pdf_file)

# Save each image to the desired directory
for i, image in enumerate(images):
    image.save(f"page_{i + 1}.png", "PNG")
```

Figure 2.4: Sample Code Snippet for Converting PDF to Image using PDF2Image library.
the most widely used topic model, however it is insufficient. As a result, OCTIS provides us with an embedding model, including CTM, ProdLDA, and NeuralLDA.
Chapter 3

Requirements

In this chapter, we will be discussing our requirements, as well as some potential requirements for considerable improvements that we might utilize.

3.1 Data Requirements

Our data collection consists of 500,000 Electronic Theses and Dissertations (ETDs) covering the time period from 1845 to 2020.

3.1.1 Dataset Cleaning

Our initial analysis of the dataset revealed that it was not clean, and had errors including improper characters, having data in the wrong fields, or not having data in important fields. Many rows had no abstracts or used placeholder words like “Abstract” or “None”. After cleaning such data, we have 333,867 rows that we can use.

3.1.2 Object Detection

Roboflow [4], a free online labeling tool, was previously used for the labeling. Twenty-four categories—such as paragraphs, figures, and tables—that were judged common in ETDs were recognized and named. We received the labeling from students in the spring semester of 2022 [9], which included both scanned and born digital ETDs. The dataset we are using
for this research has a final object count of approximately 25K pages and 100K objects.

### 3.1.3 Topic Modelling

The collection includes the title, abstract, and ETD ID, among other basic information about ETDs. As additional analyzing targets, the segmented chapters produced by the object detection algorithm in addition to the title and abstract can be used.

### 3.2 Training

We are utilizing a pre-trained object detection model, which eliminates the need for initial training data for its operation. However, to enhance the existing model’s performance, perform further fine-tuning, and improve generalization capabilities and accuracy, the model needs to be trained on a larger dataset. Model improvement is highlighted in the future scope. As for our topic modeling algorithm, it operates on an unsupervised learning paradigm, meaning there is no requirement for labeled training data in its case.

### 3.3 Usage

For inference and usage, some protocols were established with teams 5 and 6. Team 5 plays a critical role in data retrieval and storage; they provided us with efficient Read/Write APIs to fetch the raw ETD data. Once the data is in our possession, our team applies object detection and topic modeling algorithms to process it. This processed information needs to be stored back securely, for which we again rely on Team 5’s data storage APIs. On the other end, Team 6 is in charge of front-end user interactions.
3.4 Processing Requirements

Both the topic modelling and object detection tasks require significant computational power. Image processing and object detection tasks require specialized GPUs (Graphics Processing Unit), which are optimized for high-performance calculations. To meet these demands, we have access to Endeavour, a computing cluster equipped with the necessary GPUs, which serves as the backbone for our computationally intense operations.
Chapter 4

Design

This chapter will cover the aim of our project, and explain the approach and deliverables we planned to complete during the semester.

Personas

The system supports three types of personas.

1. User: Students, researchers, and other users interested to view work related to their research as well as university administration, and librarians interested in an overview of recent research in their institution.

2. Curator: Persona that can process one document at a time and view the result at the end of each process in the workflow pipeline, or can choose to batch process multiple documents at once.

3. Experimenter: Persona that can modify available model parameters and evaluate performance and efficiency of the existing system on different parameter combinations.

Aim

The aim of this work is to extend and improve upon the previous architecture (designed by Team 3, CS 5604 Fall 2022 [8]) for object detection and topic modeling. The core design and implementation of the previous architecture remain the same. The current scope of this project deals with both born-digital and scanned PDFs. The extensions to the work are detailed for object detection and topic modeling in this section.
4.1 Approach

4.1.1 Object Detection

Goals
The curator or experimenter can upload an ETD to be processed using the object detection model, and view the ETD after it has been segmented and rendered as a web page.

Previous architecture
The previous architecture is explained with respect to the user persona. The user begins by uploading an ETD file in PDF format. After the user uploads the ETD file, they can choose between two object detection models YOLOv7 and Detectron2 (based on Faster RCNN). Each page in the PDF file is converted into an image using the pdf2image module. Then, each page image is processed one by one based on the chosen object detection module. Features such as Abstract, Metadata, Table of Contents, List of Contents, and Chapters are extracted from the ETD as objects. These objects are parsed and image based objects such as figures, tables, and equations are saved to the file system and their path is returned after saving. Similarly, text objects extracted are parsed using the PyMuPDF module to extract relevant sections of text. The returned path of image objects as well as the text extracted after object parsing is stored in an XML schema. Post-processing rules are applied to the XML schema to enhance chapter segmentation and to correct false positives. The XML schema can then be used to display the ETD on the final webpage.

The architecture diagram for object detection with improvements (detailed in Section 4.2.1) is represented in Figure 4.1.
4.1.2 Topic Modeling

Goals

Users of our framework can choose topics of interest while registering to use the system. The services offered to the user include Topic List, Documents per Topic Distribution, and Similar topics.

Previous architecture

The extracted ETD text for the chosen ETD undergoes tokenization and pre-processing. Tokens are fed to topic modeling algorithms (LDA, NeuralLDA, ProdLDA, CTM, BERTopic)
to retrieve topic words and document vectors. For an ETD, related topics and documents are returned via APIs, and all of the topics produced by the model and the corresponding documents are displayed in a Topic Browser. The user can get the displayed data using Documents per Topic Distribution, Topic List, and similar topics services.

The previous architecture diagram for topic modeling is represented in Figure 4.2.

![Previous Architecture: Topic Modeling pipeline diagram](image)

Figure 4.2: Previous Architecture: Topic Modeling pipeline diagram [3]

### 4.2 Deliverables

#### 4.2.1 Object Detection

Deliverables completed

1. Establishing connection with the database/file system infrastructure managed by Team 5: The previous architecture was a stand-alone end-end application handling retrieving, uploading, and saving of ETDs and objects into a dedicated file server. This work includes establishing connection to the main file server on the Endeavour cluster handled by Team 5 which will hold all the necessary data required for any task in the
pipeline in the larger scope of the information retrieval and analysis system for ETDs. The connection is established by defining APIs for the following tasks:

- Retrieve the ETD to be processed.
- Save the image of each PDF page of the ETD.
- Retrieve the image of PDF page stored for the purpose of running Object Detection on it.
- Save the detected objects from the pages processed (text, image objects).
- Populate the ETD metadata database table with the metadata extracted from the ETD.
- Save the XML file generated for the ETD after running the object detection pipeline on it.

2. Discuss the intended UI with team 6 to display the results of the parsed ETD.

3. Add Object IDs for each object in the XML file. Team 1, working on the knowledge graph, requires an Object ID to be associated with each object stored, to be able to record structural relationships and construct an efficient and accurate knowledge graph.

4. Chapter Segmentation: The current work aims to improve the accuracy of chapter segmentation by using the page numbers from Chapters and Sections from the Table of Contents as heuristics to identify the start and end of chapters and subsections. Additionally, the page numbers extracted during object detection can be mapped to the information extracted from the Table of Contents for the segmentation process to occur during object detection thereby aiming to incorporate both processes into one as opposed to having separate architectures for the two. This heuristic approach will include two steps, parsing the PDF page by page and passing the text into an LLM (Large Language Model) such as LLaMA 2-13b in order to determine the page indices.
of the table of contents. Then, we will use a heuristic approach to parse the table of contents and extract the chapters and their corresponding page numbers. We discuss the second step further in the implementation section.

5. Implement Batch Processing: The current workflow for object detection processes only one ETD at a time, page by page. To scale the system and handle approximately 500,000 ETDs, the object detection code should be executed in batch processing mode. This effort aims to enhance the object detection pipeline by incorporating Kafka, enabling it to efficiently process batches. While preliminary experiments on identifying optimal batch sizes have been conducted, further experiments on the same with Kafka fall under the future scope.

6. Refining post-processing rules for the XML file: Building upon the post-processing rules from the previous architecture, this work aims to add or modify existing post-processing rules after implementing the proposed improvements to the object detection architecture. This is to ensure a more accurate representation of the ETD in the XML file.

7. Making Figure Captions Searchable: This work aims to accurately map the figure captions to images and make them searchable when the ETD is rendered on the webpage.

### 4.2.2 Topic Modeling

**Deliverables completed**

1. Implemented data cleaning and pre-processing steps on 500k ETDs to utilize good quality data to train LDA and BERTopic models.

2. Selected LDA model with 70 topics to be the default model being used.
3. Establishing connection with the database managed by Team 5: The previous architecture was a stand-alone end-end application handling the retrieving and saving the data into a dedicated file server. This work included establishing connection to the main file server on the Endeavour cluster handled by Team 5 which will hold all the necessary data required for any task in the pipeline in the larger scope of the information retrieval and analysis system for ETDs.

4. We have APIs suitable for Team 6 to have the UI make use of all topic modeling related functions.

5. Modified the previous architecture to make use of the BERTopic model [11] and LDA model for topic modeling. Initial inference revealed that the fixed size of 100 topics is not producing good results when we use the BERTopic model as it is not able to reduce the original 2800 topics generated for 330k ETDs down to 100 topics. Hence, we needed to improve the code implemented in previous courses. However, LDA model provided much better results.

6. Overall, make code changes to optimize processing of a large ETD dataset. This includes a data cleaning script along with pre-processing steps that need to be updated and also writing scripts to use the trained models efficiently.
Chapter 5

Implementation

5.1 Milestones

The Interim Report 1 (IR1) was the first milestone that where the team worked together to understand the objective, get familiar with the code, sort out the resources and tools available, upload the code to the Endeavour cluster, and made a semester plan to finish the whole project. Simultaneously, we also devised some improvements to extend the previous architecture for object detection and topic modeling. Interim Report (IR2) is our second milestone, where we have a working prototype satisfying the basic functionality of the code tested on 200 ETDs. As of Interim Report 3 (IR3), the API requirements for object detection have been defined, and we began integrating them into our object detection code. The Kafka framework for object detection was outlined, and tests on the chapter segmentation model were performed. Regarding topic modeling, the BERTopic model was tested on 333,867 ETDs, and we began working on implementing the LDA model to improve topic modeling results. We also started running tasks from the object detection pipeline on the set of 500,000 ETDs. The Final Report encompasses the complete code integration, clean-up, and a summary of our project, including a fully developed prototype that will be deployed in our university’s cloud.
Table 5.1: Object Detection and Chapter Segmentation Milestones

<table>
<thead>
<tr>
<th>Timeline</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Sept</td>
<td>Get familiar with the Fall 2022 Object Detection system.</td>
</tr>
<tr>
<td>8 Sept</td>
<td>Understand the YOLO V7 model.</td>
</tr>
<tr>
<td>13 Sept</td>
<td>Familiarize with the Detectron2 model implemented in the previous semester.</td>
</tr>
<tr>
<td>19 Sept</td>
<td>Explore the object detection system and set it up on the Endeavour cluster.</td>
</tr>
<tr>
<td>23 Sept</td>
<td>Perform object detection on 200 ETDs (Electronic Theses and Dissertations).</td>
</tr>
<tr>
<td>23 Sept</td>
<td>Modify the code to add (pseudo) Object IDs to the XML file.</td>
</tr>
<tr>
<td></td>
<td>Begin exploring Chapter Segmentation methodologies.</td>
</tr>
<tr>
<td>9 Oct</td>
<td>Experiment with multiprocessing to determine the optimal batch size.</td>
</tr>
<tr>
<td>13 Oct</td>
<td>Perform object detection on 200 ETDs (Electronic Theses and Dissertations) to generate XML with the IDs.</td>
</tr>
<tr>
<td></td>
<td>Test LLaMA 2 model and heuristic model for chapter segmentation.</td>
</tr>
<tr>
<td>14 Oct</td>
<td>Collaborate to plan and implement the architecture for Kafka.</td>
</tr>
<tr>
<td>19 Oct</td>
<td>Discussion with Team 1 on the XML file requirements.</td>
</tr>
<tr>
<td></td>
<td>Refine chapter segmentation code.</td>
</tr>
<tr>
<td>26 Oct</td>
<td>Discussion with Team 5 on API definitions.</td>
</tr>
<tr>
<td>7 Nov</td>
<td>Draft 1 of the APIs required for integration to the database and file system.</td>
</tr>
<tr>
<td></td>
<td>Explore RESNET for chapter segmentation.</td>
</tr>
<tr>
<td>14 Nov</td>
<td>API definition and integration.</td>
</tr>
<tr>
<td></td>
<td>Refine chapter segmentation code.</td>
</tr>
<tr>
<td>26 Nov</td>
<td>Modularize the code files for Kafka and plan to run the object detection pipeline for 500,000 ETDs.</td>
</tr>
<tr>
<td>2 Dec</td>
<td>Work on running the object detection pipeline for 500,000 ETDs.</td>
</tr>
<tr>
<td>Final</td>
<td>Documentation of work done in final report.</td>
</tr>
</tbody>
</table>

### 5.2 Object Detection

**XML Generation**

As of IR2, the object detection (YOLOv7) model has been successfully run on 200 ETDs that were uploaded to the Endeavour cluster. The object detection algorithm returns an XML file as the output of parsing the ETD. The XML schema, as seen in Figure 5.1, contains the root element as the ETD ID and three sub-elements:

1. Front: The front section includes ETD metadata, such as document title, author,
university, degree, committee, etc., as shown in Figure 5.2.

2. Body: The body contains the main ETD data, including chapters, sections, and subsection content, such as text, tables, figures, equations, and algorithms, along with the associated page numbers, as shown in Figure 5.3. Image-based objects have corresponding file paths and associated captions/numbers as seen in Figure 5.4.

3. Back: The back section provides details on the reference section of the ETD, as shown in Figure 5.5.
Figure 5.1: An example snippet of the XML schema
Figure 5.2: An example snippet of the front section of the XML schema of ETD [7]
As of IR3, it has been discussed with Team 1 who are working on the knowledge graph, to have the generated XMLs have obj_id denoting the Object ID associated with every object detected. The Object ID is received as a response from the API call after saving the objects in the database system. A snippet of the modified XML can be seen in Figure 5.6.
APIs

The main purpose of the APIs is to access the centralized resources and store intermediate results in the database as well as the file system. This enables speedy retrieval of details, be it for rendering in the front-end or for further processing tasks in the pipeline of the ETD search system. Having a centralized data store would also ensure security and authorized access to the data store.

As of IR3, the APIs required for interacting with the database and file system have been defined and provided to Team 5 for implementation, and consequently integration into the current object detection code. As a result of the object detection pipeline, the intermediary outputs (such as ETD page converted into page image, detected text-based object, etc.) would be used to populate the database tables namely the ETD_metadata, ETD_pages, and the Objects table.

The GET APIs for object detection are defined as seen in Table 5.2. These APIs are used to get the ETDs, and the saved page images of the ETD for further processing in the pipeline.
The POST APIs for object detection are defined as seen in Table 5.3. These APIs are used to populate the database tables and the file system with the intermediate results of the tasks of the object detection pipeline (such as ETD page (.pdf) to page image, objects detected in the page image, the .xml file generated for the ETD). The exact request parameters or body of the API are subject to modification based on Team 5's finalized API implementation.

<table>
<thead>
<tr>
<th>Request Method</th>
<th>Description</th>
<th>Request Parameters</th>
<th>Database/File System</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET*</td>
<td>Get ETD IDs of the .pdf file</td>
<td>-</td>
<td>Database Table: etds</td>
</tr>
<tr>
<td>GET</td>
<td>GET ETD with ETD ID</td>
<td>-</td>
<td>File System</td>
</tr>
<tr>
<td>GET</td>
<td>Retrieve page image with pageID</td>
<td>data = {</td>
<td>File System</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pageID = page ID received after</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>saving the page.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: GET APIs for Object Detection

*The implementation of this API may be temporary until the integration of Kafka, after which the ETD IDs will be available as Kafka topics.
Multiprocessing

To estimate an approximate batch size that uses anywhere between 50% - 75% of the maximum
GPU utilization resources, we made use of the Python multiprocessing and concurrent.futures library to execute multiprocessing. With a cap of maximum 4 worker nodes being utilized, we tested batch sizes of 10, 50, and 75 and monitored utilization rates using the 'nvidia-smi' command for CUDA to print an output in 1 second intervals. It was found that for a batch size of 10, the maximum resource utilization observed was 42% (Figure 5.7), for a batch size of 50, the maximum resource utilization observed was 51% (Figure 5.8), for a batch size of 75, the maximum resource utilization observed was 55% (Figure 5.9). The average resource utilization increased minimally (approximately 5%-10%) with an increase in batch size.

![Figure 5.7: Maximum resource utilization for batch size = 10](image1)

![Figure 5.8: Maximum resource utilization for batch size = 50](image2)
The experiments performed were performed on a very preliminary level and were just to get a rough estimate of the batch size we can implement for optimal batch processing. Further testing is required to fix an appropriate batch size.

**Batch Processing using Kafka**

The depicted architecture (Figure 5.10) illustrates our Kafka cluster configuration, integral to our data processing workflow. This cluster encompasses multiple Kafka Brokers, Topics, and a ZooKeeper instance for systematic coordination and configuration management. We will be using the Kafka service provided by Team 5. In this setup, the Kafka Producer operates within a Docker container, tasked with generating the IDs of the ETDs for subsequent processing.

ZooKeeper plays a pivotal role in maintaining Kafka configurations. It aids in the establishment of a Kafka Topic, where the ETD IDs are enqueued for processing. The initial processing stage involves the PDF to image conversion, for which a dedicated container continuously queries the Kafka Broker, specifically targeting the 'PDF to Image Conversion' topic. Utilizing the ETD IDs queued, the container retrieves the relevant ETDs through API calls.

Each ETD, typically in PDF format, undergoes a conversion process where every page is trans-
formed into an image format. This process meticulously accounts for the total number of pages, alongside capturing the unique ETD ID and individual page IDs. Subsequent to this conversion, these details are forwarded to the next queue in our pipeline, namely the 'Object Detection' queue.

In the object detection phase, the container, functioning as a Kafka Consumer, polls the Kafka Broker for messages on the 'Object Detection' topic. Post processing, this container transitions into a Producer role for the subsequent stage, pushing the data onto the next Kafka Topic, which is 'XML Generation'.

During the 'XML Generation' stage, the container resumes its role as a Consumer, polling the Kafka Broker for messages on the 'XML Generation' topic. The primary output from this phase is the generation of XML files, concluding the data transformation process in our Kafka-based architecture. The generated XML can be polled by consumer tasks of other teams that process the generated XML.

Figure 5.10: Current architecture for bulk processing
5.3 Heuristic Chapter Segmentation

We now move on to the segmentation of the TOC (Table of Contents). The objective is to identify the pages of the table of contents and parse them in a manner that allows us to determine where chapters begin and end.

The first algorithm, illustrated in Algorithm 1, goes a step further by using a powerful model, LLaMA 2-13b-chat-hf, to determine if a particular page (or a segment of it) belongs to the TOC. The essence of this algorithm lies in the prompt generation, which couples the extracted text with a specific question. The model is then tasked to determine if the provided text is part of the TOC. By continuously assessing pages until a negative response is received, we can efficiently bound the range of pages that make up the TOC.

The second algorithm, as detailed in Algorithm 2, is designed to parse a document’s TOC to determine chapter titles and their corresponding page numbers. By using the pymupdf library, we extract text blocks from specified pages, adjust their y-coordinates, and group lines by their y-values. Heuristics are then applied based on the indentation and presence of page numbers to build a hierarchical structure. This structure captures the chapters and their associated sub-chapters or sections.

The third approach as detailed in Algorithm 3 is to detect only numbers on the page, and then create clusters of bounding boxes for these numbers to form a column. The rightmost column is assumed to be the page numbers of the chapters or subchapters. However, this method does not differentiate between chapters and subchapters.

Together, these algorithms provide a robust and efficient method for segmenting chapters and identifying the TOC, essential to the segmentation of ETDs.
Algorithm 1 Determining Table of Contents Pages from PDF

1: procedure FindTOCPages(pdf_file)
2:     open doc from pdf_file
3:     initialize toc_pages as empty list
4:     initialize model with LLaMA 2-13b-chat-hf
5:     continuous_yes_count ← 0
6: for each page in doc do
7:     page_text ← Extract first 4000 tokens from page
8:     prompt ← CreatePrompt(page_text)
9:     answer ← model.Predict(prompt)
10:    if answer is “yes” then
11:        continuous_yes_count ← continuous_yes_count + 1
12:        Add page to toc_pages
13:    else if answer is “no” and continuous_yes_count > 0 then
14:        break▷ Stop when we get a “no” after continuous “yes” answers
15:    end if
16: end for
17: return toc_pages
18: end procedure
19: function CreatePrompt(page_text)
20:     question ← “Can you tell me if the above text is a part of the table of contents or not? Answer only with yes or no.”
21:     prompt ← “[INST] ” + page_text + ” ” + question + ” [INST]”
22: return prompt
23: end function
**Algorithm 2** Parse Table of Contents in PDF

1: procedure PARSETOC(pdf_file)
2:  initialize doc from pdf_file
3:  initialize blocks as empty list
4:  for each page in doc do
5:     for each text block in page do
6:        Adjust y-coordinates of block based on page number
7:        Add block to blocks
8:     end for
9:  end for
10: grouped_lines ← EXTRACTGROUPEDLINES(blocks)
11: hierarchy ← PROCESSLINESIMPROVED(grouped_lines)
12: cleaned_hierarchy ← CLEANHIERARCHY(hierarchy)
13: write cleaned_hierarchy to output.json
14: end procedure

15: function EXTRACTGROUPEDLINES(blocks)
16:  initialize grouped_lines as empty list
17:  for each block in blocks do
18:     Combine text spans in a line by y-coordinate
19:     Add to grouped_lines
20:  end for
21:  return grouped_lines
22: end function

23: function PROCESSLINESIMPROVED(grouped_lines)
24:  initialize hierarchy, stack as empty lists
25:  initialize prev_indent as None
26:  for each line in grouped_lines do
27:     if line contains page number then
28:        Determine indent based on x-coordinate and leading spaces
29:        Organize into hierarchy based on indent compared to prev_indent
30:        Update prev_indent
31:     end if
32:  end for
33:  return hierarchy
34: end function

35: function CLEANHIERARCHY(hierarchy)
36:  initialize cleaned as empty list
37:  for each item in hierarchy do
38:     if item has page number then
39:        Add to cleaned
40:     end if
41:  end for
42:  return cleaned
43: end function
Algorithm 3 Numeric Detection and Clustering for TOC Segmentation

1: procedure EXTRACTALLSPANSWITHBBox(pdf\_path, page\_number)  
2: \hspace{1em} Open PDF at pdf\_path and load page\_number  
3: \hspace{1em} Initialize all\_spans\_bbox as empty list  
4: \hspace{1em} for each span in each line of each block on the page do  
5: \hspace{2em} Append (span text, bbox) to all\_spans\_bbox  
6: \hspace{1em} end for  
7: \hspace{1em} Close the PDF and return all\_spans\_bbox  
8: end procedure  

9: procedure CLUSTERANDAVERAGE(y\_coords, threshold)  
10: \hspace{1em} Sort and cluster y\_coords based on threshold  
11: \hspace{1em} Calculate and return average y-coordinate for each cluster  
12: end procedure  

13: procedure CLUSTERSPANSBYUNIQUEY(spans\_bbox)  
14: \hspace{1em} Extract y-coords from spans\_bbox and find unique values  
15: \hspace{1em} Cluster y-coords using KMeans if more than one cluster  
16: \hspace{1em} Group spans by their cluster labels and return  
17: end procedure  

18: procedure EXTRACTLINESFROMCLUSTERS(clustered\_spans)  
19: \hspace{1em} Concatenate texts of spans in each cluster to form lines  
20: \hspace{1em} Return the list of formed lines  
21: end procedure
Following the application of our heuristic-based algorithms, we wanted to show some of the initial results. In Figures 5.11, 5.12, and 5.13, the results are displayed from the application of the second heuristic algorithm. This process involved initially identifying the pages of the table of contents manually and subsequently submitting them for algorithmic processing.

Figure 5.11: Input page 1 for Algorithm 2
Figure 5.12: Input page 2 for Algorithm 2
Figure 5.13: Segmented Table of Contents Output
5.4 Automated Identification of TOC Pages Using ResNet Model

Upon a detailed review of Algorithm 1, it became evident that an extensive dataset is required to effectively fine-tune the LLaMA 2 model for the accurate identification of TOC material. Additionally, the procedure proved to be time-consuming, primarily due to the substantial hardware demands associated with both inference and the fine-tuning process. Consequently, it was imperative to devise an alternative strategy that could seamlessly identify and extract TOC pages for processing, addressing these constraints.

5.4.1 Description

Instead of relying on manual identification of TOC pages or the previously mentioned approach, a new method involving a ResNet model has been developed. This model was trained on approximately 24,000 data points, consisting of 12,000 TOC page images and 12,000 non-TOC images from the ETD dataset available on ARC (Advanced Reasearch Computing) center at Virginia Tech. For TOC images, the process involved iterating through ETDs and identifying “table of contents” or “contents” pages. Conversely, non-TOC images were obtained by taking random samples from the documents.

The algorithm is designed to scan the first 50 pages of any document to identify TOC pages. The model runs for 10 iterations, with pages identified as TOC in more than 50% of the runs being selected for further processing. These pages are then subjected to the heuristic algorithm for detailed segmentation.
5.4.2 Pseudocode for ResNet Model Application

Algorithm 4 Automated Identification of TOC Pages Using ResNet

1: procedure IDENTIFYTOCPAGES(pdf_file)
2:     initialize resnet_model trained on ETD dataset
3:     initialize toc_page_candidates as empty dictionary
4:     for run from 1 to 10 do
5:         for each page in first 50 pages of pdf_file do
6:             page_image ← Convert page to image
7:             prediction ← resnet_model.PREDICT(page_image)
8:         if prediction is TOC then
9:             Increment count of page in toc_page_candidates
10:        end if
11:     end for
12: end for
13: final_toc_pages ← Filter pages in toc_page_candidates with count > 5
14: return final_toc_pages
15: end procedure

5.4.3 Evaluation and Results

The ResNet-based algorithm for identifying TOC pages was subjected to an evaluation on a segmented sample of ETDs from the dataset, comprising 56 documents. Remarkably, the algorithm achieved a 100% success rate in correctly identifying the TOC pages within these documents. This high level of accuracy underscores the effectiveness of the algorithm in handling real-world data and scenarios.
Results Visualization

The following figures, Figure 5.14 and Figure 5.15, illustrate some of the successful outcomes of the algorithm. They represent a subset of the ETDs where the TOC pages were accurately identified, demonstrating the practical utility and robustness of the proposed method.

Figure 5.14: Examples of correctly identified TOC pages
5.4.4 Application on the ETD 500 Dataset

In a further phase of evaluation, the identification model was applied to the ETD 500 dataset, encompassing 500 manually annotated ETDs. This dataset is notable for its comprehensive range of labels that categorize different sections commonly found in academic theses and dissertations. The labels include Chapters, Appendices, Reference Lists, Table of Contents, Title Pages, Abstracts, Lists of Figures, Acknowledgments, Lists of Tables, Curriculum Vitae, Dedications, Chapter Abstracts, and other miscellaneous sections. The application of our model to this diverse and annotated dataset was instrumental in assessing its capability to accurately identify and categorize various document elements, particularly focusing on the effective recognition of TOC pages.

Results of Binary Classification Approach

In an effort to streamline the classification process, we adapted our approach to convert it into a binary classification problem. This was achieved by amalgamating the labels “Table of Contents”,

Figure 5.15: Examples of correctly identified TOC pages
“List of Figures”, and “List of Tables” into a singular label termed “TOC”. Conversely, all other labels were consolidated under the “non-TOC” category. This binary categorization significantly simplified the identification process, focusing primarily on distinguishing between TOC-related content and other document elements.

Our results post this reclassification revealed notable insights into the model’s efficiency and accuracy in discerning TOC elements from non-TOC content as seen in Figure 5.16. The binary classification approach not only reduced the complexity of the task but also enhanced the model’s ability to generalize across different document structures, making it more robust in handling a variety of ETDs. The subsequent sections detail the specific outcomes and metrics achieved with this revised methodology.

Overall Metrics:
Precision: 0.8613, Recall: 0.8899, F1-Score: 0.8754

Figure 5.16: Metrics for our ResNet model on ETD 500 dataset

5.4.5 Combining ResNet Approach with Algorithm 2 for Chapter Extraction

To further enhance the efficiency of our document processing system, we integrated the ResNet-based TOC identification approach with Algorithm 2, which focuses on extracting the number of chapters and their corresponding starting pages. This combination aims to create a more streamlined and automated process for segmenting ETDs.

Success and Failure Scenarios

The integration of these methods has led to a variety of outcomes, demonstrating both successful applications and areas needing improvement.

Success Scenarios  In many instances, this combined approach successfully identified and extracted chapter information, as illustrated in Figure 5.17.
Failure Scenarios  However, there were scenarios where the heuristic algorithm failed, particularly when chapter titles were split across two lines as seen in Figure 5.19, disrupting the extraction process. An example of such a failure is depicted in Figure 5.18.

Figure 5.17: Example of a successful chapter extraction
Proposed Solutions and Ongoing Work

In response to the challenges encountered, we are currently exploring two potential solutions:

1. **Heuristic and Bounding Box Approach**: This solution involves the use of heuristics and bounding box analysis to identify titles that span multiple lines. By enhancing the algorithm’s ability to recognize and concatenate broken lines, we aim to improve the accuracy of chapter extraction.
2. BERT Model Integration: We are also experimenting with the incorporation of a BERT model to determine if consecutive lines are complementary parts of a single title. This approach leverages the model’s natural language processing capabilities to discern contextual relationships between lines.

Both solutions are currently under testing and development, with the goal of addressing the identified shortcomings and optimizing the overall performance of the document segmentation process. As part of our future work, we plan to fuse the second and third approaches into a single heuristic method. This hybrid approach aims to enhance accuracy in cases where reliable JSON objects are produced. Additionally, we intend to use the refined data from this hybrid approach to fine-tune a LLaMA 2 model, which will be designed to generate the desired outputs autonomously.

5.5 Topic Modeling

Milestones

In Table 5.4 we list out the tasks completed by the Topic Modeling team.

APIs

The main purpose of the APIs is to access the centralized resources and store intermediate results in the database as well as the file system. This enables speedy retrieval of details, be it for rendering in the front-end or for further processing tasks in the pipeline of the ETD search system. Having a centralized data store would also ensure security and authorized access to the data store.

As of IR3, the APIs required for interacting with the database and file system have been defined and provided to Team 5 for implementation, and consequently integration into the current topic modeling code. The APIs would be used to populate the database tables namely the Topic_Models, Collection_topics, and the ETD Topics table as seen in Table 5.5.

The exact request parameters or body of the API are subject to modification based on Team 5’s finalized API implementation.
<table>
<thead>
<tr>
<th>Timeline</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Sept</td>
<td>Get familiar with Fall 2022 system</td>
</tr>
<tr>
<td>8 Sept</td>
<td>Get familiar with the BERT model, OCTIS, LDA and other topic modeling approaches</td>
</tr>
<tr>
<td>13 Sept</td>
<td>Set up the code in Endeavour cluster</td>
</tr>
<tr>
<td>19 Sept</td>
<td>Train BERT model on 40k smaller ETD dataset for initial analysis</td>
</tr>
<tr>
<td>23 Sept</td>
<td>Analyze BERTopic results</td>
</tr>
<tr>
<td>23 Sept</td>
<td>Finalizing topic modeling wire frame</td>
</tr>
<tr>
<td>9 Oct</td>
<td>Train BERTopic on complete 500k ETD dataset</td>
</tr>
<tr>
<td>13 Oct</td>
<td>Analyze results of the BERT model on large dataset</td>
</tr>
<tr>
<td>14 Oct</td>
<td>Experimentation on BERT model with parameter tuning</td>
</tr>
<tr>
<td>19 Oct</td>
<td>Code refactoring and optimization for reducing training time</td>
</tr>
<tr>
<td>26 Oct</td>
<td>Writing new data cleaning and pre-processing script for BERTopic</td>
</tr>
<tr>
<td>7 Nov</td>
<td>Unsatisfactory results of BERTopic led to exploring the LDA model</td>
</tr>
<tr>
<td>14 Nov</td>
<td>Implement LDA on the 334k cleaned dataset</td>
</tr>
<tr>
<td>26 Nov</td>
<td>Improve pre-processing and experimentation of LDA model</td>
</tr>
<tr>
<td>2 Dec</td>
<td>Finalize the list of topics to be given to team 6</td>
</tr>
<tr>
<td>Final</td>
<td>Documentation of work done in final report</td>
</tr>
</tbody>
</table>

**Table 5.4: Topic Modeling Milestones**

<table>
<thead>
<tr>
<th>Request Method</th>
<th>Description</th>
<th>Request Parameters</th>
</tr>
</thead>
</table>
| POST           | Add to Topic_models to add one row for a topic modeler | data = {
name = Model name,
description= Model description,
type= Model type,
parameters= num_topics=<number of topics>,
passes= number of passes,
alpha= alpha value (eg: auto),
path= path to model
} |

**Database/File System**

- Database:
- Topic_Models table
| POST | Add to Collection_topics to store the set of topics from that modeler | data = {  
collections_id= Collection ID,  
term_list = List of terms,  
probability_list = List of probabilities  
} | Database: Collection_topics |
| POST | Topic Vector for an ETD | data = {  
etdId = ETD ID,  
topics = List of dictionaries of topic values as string and score values as float  
} | Database: ETD Topics |
| POST | Store Topic Modeling Results for ETDs | data = {  
etds_id= ETD ID,  
collections_id= Collection ID,  
topic_models_id = ID for Topic Model,  
Topic_list = List of Topics (strings) ,  
probability_list = List of probabilities  
} | Database: ETD Topics |

Table 5.5: POST APIs for Topic Modeling

Current Architecture

Figure 5.20 depicts our approach for topic analysis of ETDs using two topic modeling methods: BERTopic and LDA. Initially, the ETD data undergoes cleaning and pre-processing to prepare for
analysis. Subsequently, the refined data is processed through both BERTopic, which utilizes BERT for contextual embeddings, and LDA, a probabilistic model emphasizing word frequencies. These methods yield a comprehensive list of topics that represent the thematic essence of the documents. The derived topics are then leveraged to categorize the documents, assist in user registration by aligning user interests with the topics, and serve other downstream tasks that might benefit from a deep thematic understanding of the ETDs.
Implementing BERTopic and LDA on a large dataset of ETDs is a significant task. In our case, we
have applied LDA to 333,867 ETDs, which is a considerable volume of scholarly work. Finally, after choosing the LDA model as the default approach, assigning each ETD to one of the 73 topics formed has created a structured way to navigate and analyze this extensive collection. This approach allows for the identification of major themes and areas of research within the corpus. The decision to limit each topic to a list of 10 key words is a strategic choice that balances detail with conciseness. These keywords serve as a concise summary of each topic, making it easier to understand the primary focus of each thematic category at a glance. They can be instrumental in helping researchers and students to quickly find relevant ETDs in their area of interest.

Furthermore, this structure could facilitate comparative studies across different academic fields. By analyzing the distribution and frequency of topics across various departments or time periods, we could uncover trends in academic research, shifts in focus areas, or emerging disciplines within the scholarly community. This kind of analysis might reveal valuable insights into the evolution of research topics over time and highlight the interconnectivity of different fields of study.

**Experiments, Results and Learnings**

In our analysis of topic modeling on the cleaned ETD dataset, it is evident that LDA has shown more promising results compared to BERTopic. As illustrated in Figure 5.21, the distribution of topics generated by LDA demonstrates a more balanced and comprehensive categorization of ETDs than that of Figure 5.22, which represents the BERTopic model’s output.

A notable observation is the dominance of the “-1” topic in the BERTopic model, encompassing over 169,000 ETDs. This skewness is likely attributable to the model’s configuration, wherein the number of topics is capped at 100, a threshold that appears insufficient for the dataset’s complexity and size. BERTopic’s topic reduction technique, designed to minimize redundancy, might be overly aggressive for extensive datasets like ours, leading to an over-consolidation of diverse documents into a single, dominant topic.

Conversely, LDA’s approach seems to align better with the intricacies of the ETD dataset. LDA, a generative probabilistic model, assumes each document is a mixture of a small number of topics
and that each word’s presence is attributable to one of the document’s topics. This methodology is inherently suited for datasets with a wide range of themes and subtopics, as it allows for a more nuanced and granular topic distribution. The result is a more evenly spread representation of topics, reflecting the diverse nature of academic research covered in ETDs.

The superior performance of LDA in this context also underscores the importance of model selection based on dataset characteristics. It highlights that while advanced models like BERTopic, with their neural network foundations, offer powerful tools for natural language processing, traditional models like LDA can sometimes provide better results, especially when dealing with specialized or highly varied text corpora.

Moreover, this comparison sheds light on the need for careful tuning of model parameters. In the case of BERTopic, revisiting the topic limit and exploring ways to refine its topic reduction algorithms could potentially improve its performance for large datasets like ETDs.

Table 5.6: Summary of Topic Modeling experiments

<table>
<thead>
<tr>
<th>Experiment ID</th>
<th>Model</th>
<th>Dataset size (ETDs)</th>
<th>Stopword removal</th>
<th>Stemming</th>
<th>Lemmatization</th>
<th>Dataset cleaning</th>
<th>Number of Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BERTopic</td>
<td>40k ETDs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>BERTopic</td>
<td>524k ETDs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>BERTopic</td>
<td>334k ETDs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>BERTopic</td>
<td>334k ETDs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>500</td>
</tr>
<tr>
<td>5</td>
<td>BERTopic</td>
<td>334k ETDs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>LDA</td>
<td>334k ETDs</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>LDA</td>
<td>334k ETDs</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>LDA</td>
<td>334k ETDs</td>
<td>Yes (Custom Stopwords)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>LDA</td>
<td>334k ETDs</td>
<td>Yes (Custom Stopwords added)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>LDA</td>
<td>334k ETDs</td>
<td>Yes (Custom Stopwords added 2)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 5.6 summarizes the outcomes of nine experiments comparing the performance of two topic modeling algorithms—BERTopic and LDA—across various datasets and preprocessing techniques. The table lists experiments with BERTopic on datasets size ranging from 40k to 334k ETDs, with a number of topics set at 100, 500, and 1000, and varying levels of dataset cleaning, but without employing stopword removal, stemming, or lemmatization. In contrast, the LDA model is applied to a 334k ETDs dataset with stopword removal, and variations include stemming, lemmatization, and custom stopwords.
From our observation, the LDA model provided better results particularly in experiments 8, 9 and 10 where custom stopwords were utilized, and comprehensive data cleaning, including lemmatization, was performed. This suggests that LDA, a more traditional statistical approach for topic modeling, may benefit more from classical text preprocessing methods than BERTopic, which relies on contextual embeddings from transformer models that may not require such preprocessing to capture the semantic nuances of text.

In conclusion, our findings suggest that while both LDA and BERTopic are capable tools for topic modeling, their effectiveness is significantly influenced by the nature of the dataset and the configuration of the model parameters.

We will continue to explore additional approaches and improvements to enhance the results of topic modeling.

Figure 5.21: Distribution of documents per topic using LDA
Figure 5.22: Distribution of documents per topic using BERTopic
Chapter 6

Future Work

While Team 3 has put in a considerable amount of effort in improving the object detection and topic modeling pipelines for the information retrieval system to facilitate ETDs, there is still scope for exploration and experimentation in improving the current approach. In this section we highlight the different plans of action post-class which could be implemented in the future.

6.1 Object Detection

1. Improve the object detection model by training on a larger set of training data, and optimizing hyperparameters to improve the accuracy of the model. Additionally, improve the granularity of categories of objects extracted from the ETDs. For example, explicitly extract and store the keywords detected from the abstract of the ETD and store it along with the ETD metadata. The lower accuracy of the object detection model can be attributed to the limited number of training samples (elements such as degree, date, and algorithm have very few instances in our dataset). Furthermore, object detection models tend to struggle with localization of smaller objects such as page numbers and equation numbers.

2. Experiment with newer models such as the YOLOv8 model and chart the performance comparison between YOLOv7 and YOLOv8 for ETD data.

3. Explore a top-down object detection approach and its benefits in generating the XML for the ETD. One of the biggest challenges is that detection is not always in the top-bottom order for pages, and this necessitates the creation of an XML tree since a hierarchy of the ETD would be required for other tasks, such as segmenting an ETD into its constituent chapters,
and extracting image-based objects along with their corresponding captions. Exploring top-bottom object detection can help in easier correlation between captions and figures and tables, as well as identifying continuing paragraphs and tables that span multiple pages in the ETD.

4. Improve the performance of the object detection model with a focus on scanned PDF files. Additionally, explore data augmentation (along with applying some sort of image transformation) to the training dataset to reduce manual annotation efforts and improve model performance (especially for scanned PDFs). More can be read on this in “Analyzing and Navigating Electronic Theses and Dissertations” [3].

5. Incorporate the chapter segmentation code with the object detection pipeline to ensure accurately detected chapters in the XML file generated for the ETD. An XML with well-segmented chapters will then be passed on to Team 4 for classification and summarization tasks.

6. Set up an automated workflow to use Kafka as explained in the object detection Section 5.2 so that it can speedily handle batches. The code files for each task have been segmented but the API calls have not been defined for each task. After integrating the same, work needs to be done to set up the code to poll and publish topics from the Kafka broker and perform each task parallelly.

7. Populate both the database and the file system with the extracted data and files generated by passing the ETD through the object detection pipeline using the integrated APIs. The test_api.py file (as seen in the Developer’s Manual) holds the codes with the API definitions for object detection. The code may need to be modified slightly to fit to the final API implementation defined by Team 5. Following that, the methods can integrated into the existing code and the API calls can be initiated in the code at given placeholders (the placement of each API call in the code is given by the comments in the code files).

8. Complete processing 500,000 ETDs through the object detection pipeline and populate the database with the intermediary results of the same.

9. Clearly outline and implement the workflow for the tasks performed by each persona (the
user, the experimenter, and the curator). Ensure the smooth working and incorporation of the object detection pipeline with the front-end for each of these personas. The experiments that can be performed by the experimenter and the processing of one or more ETDs by the curator (task wise) is yet to be precisely defined.

6.2 Topic Modeling

1. For topic modeling, analyzing and improving the usage of BERTopic on the cleaned ETD dataset is needed; this is an important future goal. Integrating BERTopic for topic modeling on ETD data promises a better semantic understanding and contextual relevance, which is crucial for academic texts. It has an ability to handle large datasets efficiently and adapt to evolving research trends makes it an ideal tool for in-depth, meaningful analysis.

2. Analyze and improve the keywords formed by the LDA model by updating custom stop-words while also trying different vectorizers other than tf-idf. Further, exploring different lemmatization methods.

3. Explore other topic modeling approaches such as ProdLDA and NeurLDA on the large ETD dataset.

4. Clearly define and implement the workflow for the tasks performed by each persona (the user, the experimenter, and the curator).
Chapter 7

User Manual

7.1 Object Detection

The experimenter and the curator are the personas focused on in the user manual for object detection.

The experimenter can navigate to the front-end page to upload a PDF file for an ETD of their choice and run object detection on it. Subsequently, the experimenter will be redirected to the extracted ETD view page (HTML page). This page displays all objects extracted from the ETD in a top-to-bottom order, including metadata, chapter text, figures, captions, tables, and equations. The web layout and navigation for the same can be viewed in Team 6’s report.

The user layout for the curator is yet to be defined and is a future scope of this project. The current proposal aims to enable the curator to process more than one ETD file simultaneously. Here, we will utilize the Kafka framework, as defined in Section 5.2, to batch process the uploaded ETDs. The curator will have the option to perform object detection and generate XML files as separate tasks in the pipeline. Additional ideas regarding the different tasks the curator can perform are yet to be discussed.
7.2 Topic Modelling

7.2.1 Prerequisites

These guidelines outline the process of configuring the environment to support the operation of both the backend and frontend. In order to optimize performance, it is recommended to create a new Conda virtual environment in order to prevent any potential package conflicts. It is imperative to underscore the necessity of a Python version 3.7 or later in order to successfully install all the necessary utilities.

We can set up and activate the conda environment as follows:

1. Make sure you have Anaconda installed and ready.
2. `conda create --name <env> --file requirements_conda.txt`
3. `conda activate <env>`

By adhering to these steps, the user can make sure the environment is appropriately configured, thereby enabling the backend and frontend to function without any problem.

7.2.2 Website

The frontend user interface is composed of four main pages: the homepage, a page for searching documents, a page for exploring topics, and a page for viewing documents. The user is either an experimenter or a curator.

1. **Home Page**

   The landing page functions as the primary gateway to the website, presenting an all-encompassing synopsis of the functionalities and attributes of the system. As depicted in Figure 7.1, users can then effortlessly navigate to the document searching page and topic browsing page, which are both intended to assist them in locating and exploring relevant data in the shortest possible time.
2. **ETD Search Page**

An important component of the website, the ETD searching page enables users to browse and explore an extensive collection of documents sourced from the database. As depicted in Figure 7.2, users are able to refine their document searches using the following modifiable parameters and conduct keyword-based searches.

3. **Sorting and Filtering**

By default, search results are sorted by estimated relevance. However, users can choose to sort results by publishing date and order the results in ascending or descending order as shown in Figure 7.3. Additionally, users can filter their search results by various options generated from the config file, such as university, department, or language. This feature allows users to narrow down their search to more specific categories and find the exact documents they need.

4. **Topic Browsing Page**

As illustrated in Figure 7.4, the topic browsing page enables users to navigate and study
various subjects associated with the documents housed in the database. When a user selects a particular list of topics of interest, a page containing a collection of relevant documents will appear. Seventy topic categories are generated by default when the database contains 334,000 documents. Furthermore, in order to present the most relevant topics, the topic catalogs will be dynamically updated in response to user queries for particular subjects. Users who wish to uncover and investigate novel subjects that pertain to their areas of interest or inquiry must utilize the topic browsing page. Related documents for each topics are displayed when
clicked on a particular topic. Please refer to Figure 7.5.
The document page exhibits the entirety of the selected document’s content, in addition to important metadata. As illustrated in Figure 7.6, for a collection of text documents, the following information is required: author’s name, publication date, university affiliation, title, abstract, advisor, URL, and language. Additionally, it provides users with suggestions for relevant subjects and documents, depending on the current document’s content. This functionality facilitates effortless navigation to related subjects and documents.

When integrating the topic modeling toolkit into the larger system, it is crucial to consider both its features, and how it will interact with the system’s other components. This involves ensuring that the toolkit’s capabilities are compatible with the overall system’s requirements. Additionally,
Approximations to the MMI criterion and their effect on lattice-based MMI

ABSTRACT

Maximum mutual information (MMI) is a model selection criterion used for hidden Markov model (HMM) parameter estimation that was developed more than twenty years ago as a discriminative alternative to the maximum likelihood criterion for HMM-based speech recognition. It has been shown in the speech recognition literature that parameter estimation using the current MMI paradigm, lattice-based MMI, consistently outperforms maximum likelihood estimation, but this is at the expense of undesirable convergence properties. In particular, recognition performance is sensitive to the number of times that the iterative MMI estimation algorithm, extended Baum-Welch, is performed. In fact, too many iterations of extended Baum-Welch will lead to degraded performance, despite the fact that the MMI criterion improves at each iteration. This phenomenon is at variance with the analogous behavior of maximum likelihood estimation -- at least for the HMMs used in speech recognition -- and it has previously been attributed to `over fitting`. In this paper, we present an analysis of lattice-based MMI that demonstrates, first of all, that the asymptotic behavior of lattice-based MMI is much worse than was previously understood, i.e. it does not appear to converge at all, and, second of all, that this is not due to `over fitting`. Instead, we demonstrate that the `over fitting` phenomenon is the result of standard methodology that exacerbates the poor behavior of two key approximations in the lattice-based MMI machinery. We also demonstrate that if we modify the standard methodology to improve the validity of these approximations, then the convergence properties of lattice-based MMI become benign without sacrificing improvements to recognition accuracy.

URI

None

LANGUAGE

eng

Related Topics

Topic 86
prosodic, hdpmlm, segmentation, articulation, speech. bayesian. infants. unsupervised.

Topic 78
centering. discourse. sc, interpretation, center. pronouns. zeros. pronoun. attentional.

any UI (User Interface) provided by the toolkit should be adapted to align with the system's design language, offering a seamless and intuitive experience for both curators and end users. It's important that these UI changes do not compromise the toolkit's core functionalities, ensuring that it remains as effective and seamless as before the integration.
Chapter 8

Developer Manual

8.1 Object Detection

Conda Environment Set-Up We can set up the conda environment for the object detection pipeline by following the steps below.

1. Make sure you have Anaconda installed and ready.

2. Run `.install-structure.sh`

3. Run `.install-dependencies.sh`

4. Create a new environment with the `.yml` file

5. Check if the latest version of Torch is installed

6. Run `pip install torchvision`

7. Run `apt-get update && apt-get install ffmpeg libsm6 libxext6 -y`

8. Run `pip install pdf2image`

9. Run `conda install -c conda-forge poppler`

config.yml The `config.yml` in the config folder is used to configure the relative paths of folders in the Endeavour cluster as shown in Figure 8.1. If the generated intermediary outputs need to be stored in the local file system, the relative paths can be rewritten here.
Python Code Files All the code files are accessible through Team 3’s container on the Endeavour cluster [1]. Figure 8.2 shows the directory containing all the code folders and some of the code files required for the object detection pipeline.

The flask_app folder as shown in Figure 8.3, contains 200etdsscript.py, the executable which takes in the path to the folder containing ETD files and runs the object detection pipeline by invoking

Figure 8.1: config.yml file

Figure 8.2: Team 3 container on the Endeavour cluster
the task method from the flask_main.py.

The xml_parser as shown in Figure 8.4, contains the xml_utils.py and the pdf_utils.py which are auxiliary files crucial to the tasks of XML generation and conversion of ETD pages into images. The layout_models folder contains the YOLOv7 model along with files containing the model training weights, and a file initializing the model. The test_api.py holds definitions for each of the APIs for the object detection pipeline and need to be invoked in the flask_main.py file as well as in xml_util.py once implemented to populate the database with the details of the ETD.

8.2 TOC Segmentation

In order to run the table of contents segmentation algorithm, navigate to the main project folder’s directory and run the script of resnet.py. If the model is further trained for better image classification, then simply replace the resnet_toc_classifier2.pth with the new model weights, then rename

Figure 8.3: flask_app

Figure 8.4: xml_parser
the file path in the code in the part mentioned in Figure 8.5.

```python
# Load the trained model
model_path = "resnet_toc_classifier2.pth"
model = models.resnet50(pretrained=False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)
model.load_state_dict(torch.load(model_path))
model = model.to(device)
model.eval()
```

Figure 8.5: Team 3 container on the Endeavour cluster with model path

Run the script by using the Python command on the script “resnet.py”. The script will run on the default directory for the ETDs that needs to be processed. However, if that path needs to be changed, it can be changed at the end of the script by replacing the main directory that contains the ETDs as shown in Figure 8.6.

```python
root_directory = "/mnt/raw_ETD/segmentedETDs"  # starting directory
process_pdfss_in_directory(root_directory)
```

Figure 8.6: Team 3 container on the Endeavour cluster with root directory

## 8.3 Topic Modeling

### 8.3.1 Prerequisites

These guidelines outline the process of configuring the environment to support the operation of both the back-end and front-end. In order to optimize performance, it is recommended to create a new Conda virtual environment in order to prevent any potential package conflicts. It is imperative to underscore the necessity of a Python version 3.7 or later in order to successfully install all the necessary utilities.

We can set up and activate the conda environment as follows:

1. Make sure you have Anaconda installed and ready.
2. conda create --name <env> --file requirements_conda.txt

3. conda activate <env>

By adhering to these steps, the user can make sure the environment is appropriately configured, thereby enabling the back-end and front-end to function without any problem.

In this section, we will provide steps to implement the topic modeling scripts and also use the trained LDA model. Click on the links to access files in the Endeavour cluster.

Refer to Figure 8.7 for the topic modeling folder structure. Figure 5.20 gives an idea of the topic modeling flow that is used in the scope of this project.

To reach the topic modeling repository, refer to code in the Endeavour cluster [2].

1. First, run the topicmodeling.yml file to set up the conda environment for topic modeling.
   You can also refer to the README.txt file to set up your environment.

2. Make changes in the config.ini to either train a model or load a model by changing the arguments.

3. The config.ini is also used to pass the path to the database.

4. Following this, make changes in /team3/fall2023/TopicModelling/
   DocumentBrowsingToolkit-main/back_end/server.py file to either run the BERTopic model or the LDA model.

5. To access already saved models, please refer to /team3/fall2023/TopicModelling/
   DocumentBrowsingToolkit-main/back_end/
   TopicModelingKit/src/models/bertopic_model_bk file for the BERT model or /team3/fall2023/
   TopicModelling/DocumentBrowsingToolkit-main/back_end/
   TopicModelingKit/src/models/LDA_model for the LDA model.

6. To analyze and use the saved LDA and BERTopic models, go to /team3/fall2023/TopicModelling/
   DocumentBrowsingToolkit-main/back_end/TopicModelingKit/TopicModeling scripts/
   Topic Modeling scripts.
7. The `test_LDA.py` file contains APIs to POST into the database tables with topic modeling results.

![File tree]

Figure 8.7: Team 3 container on the Endeavour cluster for Topic Modeling.

The outputs are stored in CSV, JSON and TSV folders and have been passed along to other teams. The code for generating outputs can be found in the topic modeling scripts.

In summary, this section outlines a guide for implementing topic modeling using LDA and BERTopic models within a specified project scope. It provides steps for setting up the environment, configuring models, and integrating them with a database, as well as accessing and utilizing saved models. The process culminates in generating outputs in various formats, facilitating collaboration and data sharing with other teams.
Bibliography


