

# Final Report

## CS 5604: Information Storage and Retrieval

Team 4: Language Models, Classification, Summarization & Segmentation  
Kaushik Ganesan, Dharneeshkar Jayaprakash, Deepak Nanjundan, Abhilash Neog, Aditya  
Shah, Deval Srivastava

Instructed by Professor Edward A. Fox  
SMEs: Bipasha Banerjee, Sara Ahmadi

January 11, 2023  
Virginia Polytechnic Institute and State University  
Blacksburg, Virginia 24061

## **Abstract**

Under the guidance of Dr. Edward A. Fox, the class of CS5604 Fall 2022 semester at Virginia Tech was assigned the task of building an Information Retrieval and Analysis System that can support a collection of at least 5000 Electronic Theses and Dissertations (ETDs). The system would act as a search engine, supporting a number of features, such as searching, providing recommendations, ranking search results, and browsing. In order to achieve this, the class was divided into five teams, each assigned separate tasks with the intent of collaborating through CI/CD. The roles can be described as follows: Content and Representation, End-user Recommendation and Search, Object Detection and Topic Models, Classification and Summarization with Language Models, and Integration and Coordination. The intent of this report is to outline the contribution of Team 4, which focuses on language models, classification, summarization, and segmentation. In this project, Team 4 was successful in reproducing Akbar Javaid Manzoor's pipeline to segment ETDs into chapters, summarize the segmented chapters using extractive and abstractive summarizing techniques, and classify the chapters using deep learning and language models. Using the APIs developed by Team 1, Team 4 was also tasked with storing the outcomes of 5000 ETDs in the file system and database. Team 4 containerized the services and assisted Team 5 with workflow automation to help automate the services. The project's main lessons were effective team collaboration, efficient code maintenance, containerization of services, upkeep of a CI/CD workflow, and finally effective information storage and retrieval at scale. The report describes the goals, tasks, and achievements, along with our coordination with the other teams in completing the higher-level tasks concerning the entire project.

# Contents

<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Overview</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Discussions with Subject Matter Experts . . . . .	1
<b>2 Review of Literature</b>	<b>3</b>
2.1 Segmentation . . . . .	3
2.1.1 Segmenting Electronic Theses and Dissertations By Chapters . . . . .	3
2.1.2 Chapter Captor: Text Segmentation in Novels . . . . .	3
2.2 Summarization . . . . .	3
2.3 Classification . . . . .	4
2.3.1 Multi-label Classification for Documents . . . . .	5
2.3.2 Text Classification using BERT . . . . .	5
2.3.3 Long Text Classification using Doc2Vec . . . . .	5
2.3.4 Long Text Classification using Transformer Based Networks . . . . .	6
2.3.5 SciBERT . . . . .	6
2.4 Knowledge Distillation . . . . .	7
<b>3 Requirements</b>	<b>8</b>
3.1 Overall Project Requirements . . . . .	8
3.2 Team 4 Requirements . . . . .	9
3.3 Collaboration . . . . .	9
<b>4 Design</b>	<b>10</b>

4.1	Overall Flow . . . . .	10
4.2	Segmentation . . . . .	11
4.2.1	Approach . . . . .	11
4.2.2	Methodology . . . . .	11
4.2.3	Evaluation . . . . .	12
4.2.4	Future Work . . . . .	13
4.3	PDF Parsing and Cleaning . . . . .	14
4.4	Summarization . . . . .	14
4.4.1	Approach . . . . .	14
4.4.2	Methodology . . . . .	15
4.4.2.1	Summarization using Extractive models . . . . .	16
4.4.2.2	Summarization using Abstractive models . . . . .	16
4.4.3	Challenges . . . . .	16
4.4.4	Future work . . . . .	17
4.5	Classification . . . . .	17
4.5.1	Approach . . . . .	17
4.5.2	Chapter Classification: Methodology . . . . .	18
4.5.2.1	Data Collection and Preprocessing . . . . .	19
4.5.2.2	Chapter Classification using BERT and SciBERT . . . . .	19
4.5.2.3	Chapter Classification using Longformer . . . . .	20
4.5.2.4	Chapter Classification based on chapter summary . . . . .	21
4.5.2.5	Chapter Classification using SVM and Random Forest . . . . .	21
4.5.3	Model Evaluation . . . . .	22
4.5.4	Inference on ETD subset . . . . .	23
4.5.5	Future Work . . . . .	23
4.5.5.1	Metadata Classification . . . . .	23
4.5.5.1.1	Methodology . . . . .	23
4.5.5.1.2	BERT-based Metadata Classification . . . . .	24

4.5.5.2	Knowledge Distillation . . . . .	25
4.5.5.3	Citation Intent Classification . . . . .	25
4.5.5.4	Investigating Chapter Lengths . . . . .	25
4.6	User Interface . . . . .	26
4.7	Tools . . . . .	28
<b>5</b>	<b>Implementation</b>	<b>29</b>
5.1	Tasks and Timeline . . . . .	29
5.1.1	Segmentation Tasks and Timeline . . . . .	29
5.1.2	Summarization Tasks and Timeline . . . . .	30
5.1.3	Classification Tasks and Timeline . . . . .	30
<b>6</b>	<b>User Manual</b>	<b>31</b>
6.1	Experimenter Page . . . . .	31
6.2	Future Work . . . . .	35
<b>7</b>	<b>Developer Manual</b>	<b>36</b>
7.1	Installation Setup . . . . .	36
7.2	APIs . . . . .	37
7.3	Instructions . . . . .	38
7.3.1	Classification . . . . .	38
7.3.2	Summarization . . . . .	39
7.3.3	Summarization API Driver . . . . .	41
7.3.4	Segmentation . . . . .	41
7.3.5	Parsing and Cleaning . . . . .	42
7.4	Workflow Automation APIs . . . . .	43
	<b>Bibliography</b>	<b>45</b>

# List of Figures

4.1	Overall flow . . . . .	10
4.2	High-level view of Segmentation task . . . . .	11
4.3	Summarization flow diagram . . . . .	14
4.4	Chapter summary JSON . . . . .	15
4.5	Classification flow diagram for chapter classification . . . . .	18
4.6	Chapter Classification using Longformer . . . . .	20
4.7	Chapter Classification based on summaries . . . . .	21
4.8	Classification using SVM and Random Forest . . . . .	21
4.9	Metadata classification using BERT . . . . .	24
4.10	Experimenter User Interface wire-frame . . . . .	27
6.1	Experimenter User Interface web page . . . . .	31
6.2	UI displaying segmented chapters and summaries . . . . .	32
6.3	Dropdown box to select summarization model . . . . .	32
6.4	Dropdown box to select classification model . . . . .	33
7.1	Summarization Internal Config . . . . .	40

# List of Tables

4.1	Evaluation for Segmentation Model . . . . .	13
4.2	Evaluation of classification models . . . . .	23
5.1	Tasks and Timeline for Segmentation . . . . .	29
5.2	Tasks and Timeline for Summarization . . . . .	30
5.3	Tasks and Timeline for Classification . . . . .	30
6.1	Description of video team4_experimenter_demo.mp4 . . . . .	34
6.2	Description of video team4_documentView_demo.mp4 . . . . .	34

# List of Abbreviations

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CI/CD	Continuous Integration and Continuous Deployment
ETDs	Electronic Theses and Dissertations
HTML	HyperText Markup Language
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PDF	Portable Document Format
SVM	Support Vector Machine
UI	User Interface
XML	Extensible Markup Language



# Chapter 1

## Overview

### 1.1 Introduction

As part of this project, we were tasked with building a state-of-the-art Information Retrieval and Analysis system that functions effectively and reliably for over 5000 ETDs. In order to accomplish this, Team 4 undertook the role of working and experimenting with language models. This primarily involved segmentation, summarization, and classification of ETDs. One of our primary roles was also to collaborate with Team 2 (which is responsible for designing the end-user recommendation and search system), by building an end-to-end segmentation pipeline, with output generation in the format required for front-end representation. In order to accomplish this, we explored the existing datasets and examined ways we can segment the ETDs into chapters. Given an ETD PDF as an input, the ideal output would be a set of PDFs, corresponding to each chapter of the ETD.

Along with this, we were also responsible for building a summarization system. As part of this work, we generated three different summaries using different models and chose the best one by qualitatively evaluating the summaries while also considering the time constraints to run the model. Our third role involved chapter classification. Based on the performance achieved, we have proceeded with the most suitable method for the classification of ETDs. Finally, we built a front-end application that allows users to run and experiment with the trained models, through a user interface.

### 1.2 Discussions with Subject Matter Experts

Before settling on our individual objectives and deadlines, we spoke with Bipasha and Sara, our Subject Matter Experts on the Classification, Segmentation, and Summarization of ETDs. In addition to the expectations of the end users who may use our system in the future, we also got an overview of a few general facts about the dataset.

Around 500,000 ETDs from more than 42 universities have been gathered by the team working with Dr. Fox. The collection of ETDs spans a large number of domains and a considerable amount of time (from 1875 to 2022).

The two main categories of ETDs that were gathered are:

- Born-Digital ETDs: These are ETDs that were uploaded digitally to university repositories. The majority of the new ETDs are digitally born.
- Non-Born-Digital ETDs: ETDs that weren't originally submitted digitally but were later scanned and submitted. Some older ETDs weren't always digital. A few of the scanned ETDs have also had their text extracted using OCR (Optical Character Recognition). However, there are issues with the recovered text that OCR produced.

Earlier work on chapter segmentation was recommended to us as a starting point, and we were encouraged to extend it to segment ETDs into chapters. Additionally, we were advised to use pre-trained versions of models such as Longformer [10] to summarize the chapters of the ETDs and establish a baseline for future works. It was also suggested that we build a benchmark dataset to assess the summaries produced by the models. If the ETDs are associated with journal papers, summaries of the ETDs can be found in the related publications' abstracts. To acquire a background on the classification tasks to be completed, it was suggested that we refer to Palakh's thesis [26].

# Chapter 2

## Review of Literature

### 2.1 Segmentation

#### 2.1.1 Segmenting Electronic Theses and Dissertations By Chapters

In his work [29], Javaid has discussed a method for segmenting a born-digital ETD PDF into chapters. A labeled dataset is required to train any machine learning model to automatically partition large publications into chapters. However, there are no datasets that include ETDs as well as information about the underlying document structures. Therefore, arXiv’s repository [30] is used to create a dataset that contains 1459 ETDs’ segmented chapters. This is accomplished by altering the LaTeX files according to chapter boundaries. The dataset obtained is used to train a deep learning model, which may then be used to segment other born-digital ETDs.

#### 2.1.2 Chapter Captor: Text Segmentation in Novels

This paper [4] explores a hybrid approach combining BERT inference and Regex rule matching for chapter segmentation. The dataset used for training the BERT model consists of 8400 English fiction books. Initially, when a text is given as input, the custom BERT model tokenizes the text, and using the sliding window approach it outputs the confidence score for each token. Max pooling is performed to get a single confidence score for each token. Then, de-tokenization is performed, and potential headers are extracted. The output is next given to the Regex rule matcher where they have a list of the common chapter heading keywords. This rule matcher tries to find the chapters that the BERT model misses. Finally, the chapter outputs are refined and each chapter is saved in a separate XML file.

### 2.2 Summarization

Text summarization is the process of converting a large body of text into smaller yet concise text containing the key points and the overall meaning of the original document. Though

text summarization started gaining attention as early as the 1950s, with the rapid growth of textual data and large documents like ETDs in recent times, it has become even more imperative to summarize large volumes of text for better navigation and retrieval of relevant information. Text summaries not only reduce reading time but also improve the effectiveness of indexing [15].

Generally, there are two approaches to text summarization – Abstractive and Extractive Summarization. Abstractive Summarization involves generating completely new text data to capture the meaning of the source [16]. Extractive Summarization, on the other hand, aims to select the most relevant (to the meaning of the document) phrases and sentences within the source to create the new summary [16]. Although Abstractive Summarization generates more human-like and potentially optimal solutions, it is quite challenging. While most of the works in the past have mainly focused on Extractive Summarization, the emergence of deep learning has greatly advanced the progress of Abstractive Summarization. Nallapati et al. [17] in 2016 introduced a sequence-to-sequence model with attention to solve the problem of Abstractive Summarization. They proposed a model that can freely read and generate text and achieved state-of-the-art performance in two different English corpora. Although this led to a rise in research involving sequence-to-sequence models, those generally suffered from two limitations – inaccurate and redundant information generation [18]. To address these limitations, See et al. [18] proposed a modified sequence-to-sequence model with a hybrid pointer-generator network and coverage mechanism. The pointer generator network consists of a pointer that can copy text from the source while retaining the capability to generate new text using the generator. The coverage technique helps keep track of the text already summarized, thus reducing redundancy.

Our problem area involves text summarization of Electronic Theses and Dissertations (ETDs). Previously, Ingram et al. [19] applied deep learning techniques to summarize ETDs. Their work makes use of transfer learning (with standard models for Abstractive Summarization) to address the issue of lack of ETD training data. In the current project, we explored this idea, however, owing to certain constraints, stuck to pre-trained transformer models.

## 2.3 Classification

Classification is one of the major tasks that can be performed on text. When we classify a piece of text, we assign it certain labels, that in some way describe the document. In classification terminology, we can either assign a single label or multiple labels to text input. In our research, we will be performing multi-label classification over documents. Specifically, we will be classifying long text pieces which can be either chapters or documents. In the next section, we will outline some of the previous work that has been done in this space.

## 2.3. CLASSIFICATION

### 2.3.1 Multi-label Classification for Documents

Since each chapter could be associated with several labels, the chapter-level classification we perform in this work is multi-label in nature. An ETD may fall under more than one discipline. This is confirmed by the practice of ProQuest (we have used the dataset of ProQuest – now a part of Clarivate – to create the initial baseline), which enables authors to pick numerous secondary categories in addition to a primary category.

The majority of conventional algorithms are designed for classification with a single label. Therefore, the multi-label classification problem can be split up into several single-label classification problems in order to use the pre-existing algorithms. Some of the common techniques used to convert multi-label classification problems to multiple single-label classification problems are (i) the **OneVsRest** method and (ii) the **Label Powerset** method. The **OneVsRest** method builds numerous independent binary classifiers, and when a test instance that has not yet been observed needs to be classified, the class labels can be chosen from among those that have the highest confidence. The underlying correlation between the class labels is ignored in this method [31]. In the **label powerset** approach, each label combination is treated as a separate class, converting a multi-label dataset into a single multi-class dataset. By adding an instance to a class made up of a number of labels, it is capable of performing multi-label classification [31].

### 2.3.2 Text Classification using BERT

Since the seminal paper from Vaswani et al. [14], proposing the transformer architecture, the NLP landscape has changed and current methods have leapfrogged in performance. The same can be said for text classification approaches. Before the development of transformer-based methods, researchers would use Word2Vec [5], etc. to generate representations for text and then apply classification methods to it. Now BERT [9] can be used to classify text pieces effectively by first generating a contextual representation of the input text and then applying a logistic regression style classifier on top of it. In studies, it has been shown that BERT models are able to outperform previously known best methods. Inspired by this progress, we also experiment with BERT-based methods.

### 2.3.3 Long Text Classification using Doc2Vec

A few machine learning algorithms exist that convert words into numeric vector representations. Some well-known ones include Word2Vec [5] and GloVe [6]. However, these are insufficient for paragraph-level or document-level representation. Lee et al. [12] proposed generating paragraph vectors and an unsupervised algorithm that generates fixed-length representations from the variable-length text. Their experiments show that this method is very effective in classifying text documents. To do that you generate the embeddings and then

send them to a logistic regression-style classifier.

### 2.3.4 Long Text Classification using Transformer Based Networks

In a prior section, we discuss BERT for text classification. A limiting factor in this method is the max sequence length of 512 tokens in BERT and similar models. This implies that most documents will have to be truncated, leading to a loss of information. Hierarchical transformers [8] attempt to address this issue by incorporating either a recurrence unit or a transformer layer to obtain the BERT representation of input by splitting it into shorter segments. This method performed better than many of the approaches proposed until then. Apart from this, Park et al. [2] first selected the most important sentences from the long text using TextRank [20] and then applied BERT style classification on the reduced text. This method proved effective in their testing. More recently advancements have been made in improving attention methods. Longformer [10] introduces an attention mechanism that scales linearly. Due to this advancement, Longformer is able to accept texts of lengths up to 4096 tokens, which allows it to retain significantly more information compared to BERT; it performs well on long text classification. Another work, BigBird [11], also introduces a new sparse attention mechanism and accepts texts of lengths up to 4096 tokens. This method also does well on long text classification.

### 2.3.5 SciBERT

As per “SciBERT: A Pretrained Language Model for Scientific Text” by Iz Beltagy et al. [32], SciBERT is a language model that has been trained on various scientific corpora to carry out various downstream scientific NLP tasks. Sequence tagging, sentence categorization, dependency parsing, and many other tasks are among them. On a significant subset of these downstream tasks, SciBERT has produced brand-new state-of-the-art outcomes. The BERT architecture is the foundation of SciBERT. The only difference between it and BERT is that it has been pretrained on a corpus of scientific text. Tokenizing input text and creating a model’s vocabulary are both performed by BERT using WordPiece [9] tokenization. The vocabulary includes the most common word and subword combinations. Tokenization and vocabulary generation for SciBERT are done using a lexicon of scientific words called SciVOCAB [32]. SciBERT was trained using 1.14 million papers from Semantic Scholar [52]. The papers’ complete texts, including their abstracts, are used. The papers are typically 154 sentences long. The results of SciBERT are very encouraging. SciBERT has outperformed BERT on seven biomedical datasets. It has performed about the same as BioBERT [53] on biomedical datasets. SciBERT surpassed BERT and attained state-of-the-art outcomes on all five datasets tested by the authors for computer science and other disciplines.

## 2.4 Knowledge Distillation

Deep learning models have been performing better and better on many tasks, but alongside the increase in performance, the computational cost required to run these models has also grown a lot. A few methods have been designed to optimize the models and reduce the costs, Knowledge Distillation being one of them. Initially proposed by Hinton et al. [7], this method involves firstly training a large model with a huge number of parameters on the target dataset. Once this model has been trained it's called the teacher model. Now, a smaller model with fewer parameters is designed. This model will be called the student model. The student model is trained in such a manner that it learns to replicate the results of the parent model. Taking an example of classification we would compute the class probabilities for every item in the dataset using the teacher model and then use these probabilities as the target for the student model. DistillBERT [13] is the distilled version of the BERT model, which has 40% fewer parameters when compared to the teacher BERT model.

# Chapter 3

## Requirements

### 3.1 Overall Project Requirements

As the overarching goal of the project is to build a state-of-the-art information retrieval and analysis system, the main requirements can be listed as follows:

- Either a complete document or a sub-document derived from another document through tasks such as segmentation or extraction should be used for processing.
- Both the metadata as well as the entire text should support searching.
- Any aspects related to information derived from both the data and metadata should dictate the searching and browsing functionalities.
- Any aspects related to information derived from documents – through processing techniques such as summarization, classification, and analysis – should dictate the searching and browsing functionalities.
- Actions such as clicks and user queries should be logged as well as analyzed to increase user support. User-provided recommendations and feedback should also be considered and implemented.
- Techniques such as weighting or indexing should be selected in a manner that ensures the effectiveness of operations.
- The most effective method should be used to rank search results.
- The content collection should dictate how pre-processing is handled, for both the managing of linguistic issues (e.g., word sense disambiguation, lemmatization, sentence identification, and object recognition), as well as for managing input formats (e.g., Text, PDF, JSON).
- In order to ensure that further progress and improvements can be made, any produced software and data should be released to all members of the project.



### 3.2 Team 4 Requirements

As our team acts as experimenters with machine learning based on language models, our expected tasks can be outlined as follows:

- Developing a front-end application to access our services and run the models through a UI.
- Building an end-to-end system for segmentation – implement Javaid’s segmentation model into the general front-end layout in collaboration with Team 2, and compare this implementation with Team 3’s segmentation technique.
- Implementing a PDF parsing and clean-text generating script as a standalone service.
- Building an end-to-end system for summarization.
- Performing chapter classification by experimenting with language models to get the discipline labels.

### 3.3 Collaboration

The other teams collaborated with us as follows.

- Team 1: Finalized API contracts. Collaborated to run our clean and parse script, as well as the segmentation pipeline in their container.
- Team 2: Developed front-end (experimenter and document view pages). Used chapter summaries for indexing.
- Team 3: They provided digital objects and XML files that could be used for classification and summarization.
- Team 5: GPU setup for training, containerizing our services, CI/CD, and workflow automation.

# Chapter 4

## Design

### 4.1 Overall Flow

The key aim of the project is to segment, summarize, and classify a subset of the ETD dataset. Figure 4.1 shows the overall flow diagram of the system. For this project, only born-digital documents from the ETD dataset were considered. This is because Team 4's segmentation pipeline only works with born-digital type documents. Initially, the digital ETD document is given as an input to the segmentation pipeline and the segmented chapter output is generated. These generated chapters are passed into summarization and classification pipelines. Finally, after post-processing, all outputs are integrated into the front-end of the system.

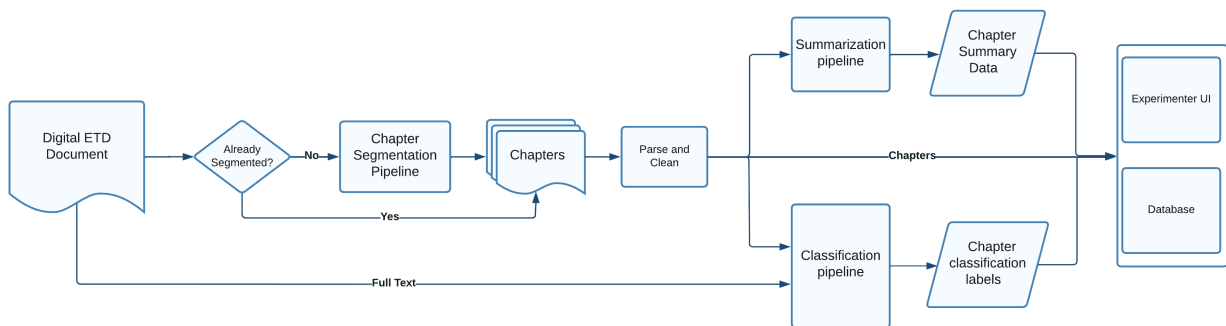


Figure 4.1: Overall flow

In the sections below we elaborate further on the functioning and implementation of each component defined within the overall flow.

## 4.2 Segmentation

### 4.2.1 Approach

The segmentation task is performed to divide the ETD into chapters. The generated segmented chapters can then be given as input to services for chapter summaries, chapter topics, and chapter searches.

Our approach was to build on the M.S. thesis work of Akbar Javaid Manzoor, which was one approach to ETD segmentation into chapters. His work is summarized as follows. Initially, Javaid’s segmentation pipeline was implemented using two datasets [29]. The main dataset contains 1459 ETD documents in LaTeX format which were obtained from the open-access arXiv repository. The second dataset containing 150 ETD documents with manually labeled chapter boundaries.

The pipeline consists of two parts. In the first part, the main .tex file is identified for documents in the first dataset. Then, the chapter start is found and delimiters are inserted into the chapter start location. Additionally, the chapters are saved as separate PDF files. In the second part, the generated chapter PDFs are used as ground truth to train a deep learning model. This approach is useful for scanned and born-digital documents, but does better with the latter (according to an evaluation with the second dataset).

### 4.2.2 Methodology

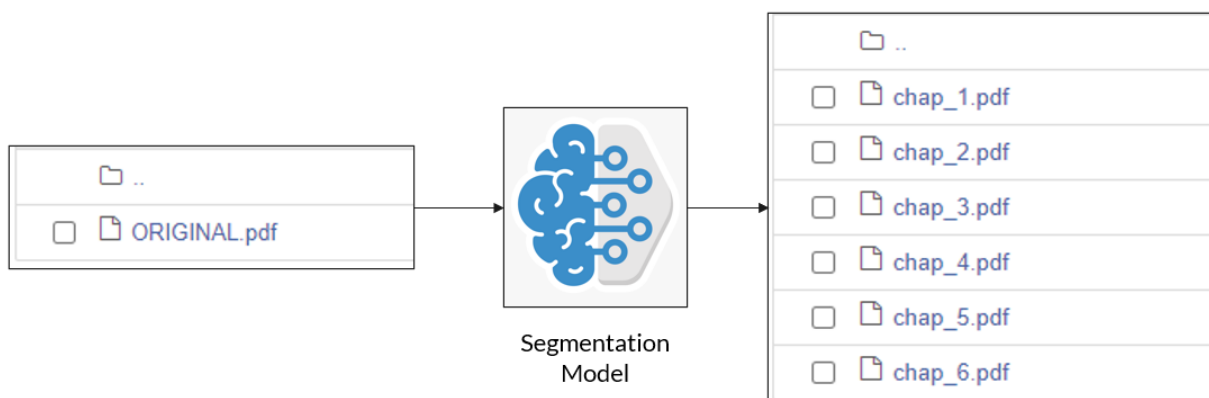


Figure 4.2: High-level view of Segmentation task

The high-level view of the segmentation module is shown in Figure 4.2. When a PDF is given as input to the segmentation pipeline, the model predicts the chapter boundaries and saves each chapter as a separate PDF.

First, Javaid’s latest deep learning model checkpoint is loaded, and the model architecture is initialized. The model consists of two networks that try to extract both the image and the text features from each page and predict a classification label (0: “FM PAGE”, 1: “FM START”, 2: “CH START”, 4: “CH PAGE”, 3: “EM START”, 5: “EM PAGE” ). The predicted labels for each page are stored in an array. Next, the ideal way to extract the chapter boundaries from the prediction array is to traverse the array and find the page numbers from one “CH START” to the next one. Also, we check if the pages in between two “CH START” occurrences are “CH PAGE”. But this approach is over-dependent on the “CH START” label. If one “CH START” prediction goes wrong the entire chapter is not detected. When testing this approach on more than 100 ETDs, an average of only 2 chapter PDFs were saved. Due to the above shortcomings, we came up with a slightly modified approach to extract chapter boundaries.

In this approach, we leverage the “CH PAGE” information. From Javaid’s model metrics, it’s clear that “CH PAGE” has a very good F1 score when compared to “CH START”. The prediction array is traversed and subarrays with more than 6 consecutive “CH PAGE” entries are saved as a chapter. The start and end of the subarray’s labels are checked and a few other edge cases are taken into consideration. Finally, this new approach is also tested on more than 100 ETDs and an average of 4 chapters PDFs are detected. Therefore, the new approach has a better performance, comparatively.

Finally, the segmentation code is packaged with all requirements and model weights. The packaged service is moved to the Team 1 container. This makes it easier to access the ETD data and also saves a few API calls. The segmentation service is then integrated with Team 1’s database and file system saving functionality. The segmentation is run on the 5000 born-digital ETDs subset. All ETDs in this subset belongs to the “000” folder. The generated chapter PDFs are saved in a folder named “Team4\_chapters” inside each corresponding ETD folder, and entries are made in the objects table.

### 4.2.3 Evaluation

The segmentation model is evaluated and the results are shown in Table 4.1. The model is evaluated on a test set created by Javaid with 100 sample data. We have also used Javaid’s evaluation script to calculate the Precision, Recall, and F1 scores for all 6 classes. The “CH Page” class has the best F1 score.

## 4.2. SEGMENTATION

Table 4.1: Evaluation for Segmentation Model

Page Type	Precision	Recall	F1 Score
FM Start	64%	47%	54%
FM Page	65%	44%	52%
Chapter Start	82%	77%	79%
Chapter Page	91%	97%	94%
EM Start	78%	68%	73%
EM Page	83%	64%	72%

### 4.2.4 Future Work

We recommend additional work on chapter segmentation. We suggest testing three different approaches, evaluating the results, and then tuning the best of the methods.

The first and easiest new method would be to use an updated version of Javaid’s segmentation software. After he provided the software that was used during the semester, he continued work, and developed an improved deep learning model. That is likely to give better results than the earlier version.

The second method would build on the work of another class team. Team 3 implemented an object detection-based approach to segment ETDs into parts, such as chapters. This approach is useful for both scanned and born-digital documents. This can be leveraged to perform summarization for scanned as well as born-digital documents.

Finally, chapter segmentation can be performed using the Chapter Captor [4] approach. The first step for this approach is to extract the text from the PDF. Tools like PDFMiner, PyPDF2, and PyMuPDF can be used to convert the PDF to an HTML or XML format. The extracted text is then given as input to the BERT model. The BERT model predicts the chapter headers. Furthermore, regex rules are applied to find the chapter headers that the BERT model misses out on. Finally, based on the predicted chapter headers, unique delimiters are inserted which serve as positional information for chapters. Furthermore, to improve the model performance, the dataset size can be increased and extensive model tuning can be performed.

Regarding determining the best approach, evaluation is essential. For any system, testing is a crucial task. When it comes to machine learning systems, it’s a bit trickier to do testing. One good approach is to first validate the traditional software codes surrounding the machine learning model which includes input data, data preparation, feature engineering, configurations, and pipeline integration.

## 4.3 PDF Parsing and Cleaning

Segmented chapter PDFs are required to be parsed and converted into text files before utilizing them for downstream tasks like summarization and classification. Hence, instead of performing PDF parsing and text cleaning at every stage, this task has been separated out, to be operated independently, such that this service gets called every time after segmentation but before the generated clean text files are stored for the other services.

In the current implementation, the parsed PDF is cleaned by removing only the figures and tables. It is, however, desirable to remove equations from the text as well. This would be an important part of future work.

## 4.4 Summarization

### 4.4.1 Approach

ETD text summarization is a challenging task primarily due to the size of the input text, and the lack of suitable training data. As reported by [26], summarizing the complete ETD results in a loss of key concepts present within the chapters. Hence, the proposed methodology deals with summarization at the chapter level as shown in Figure 4.3.

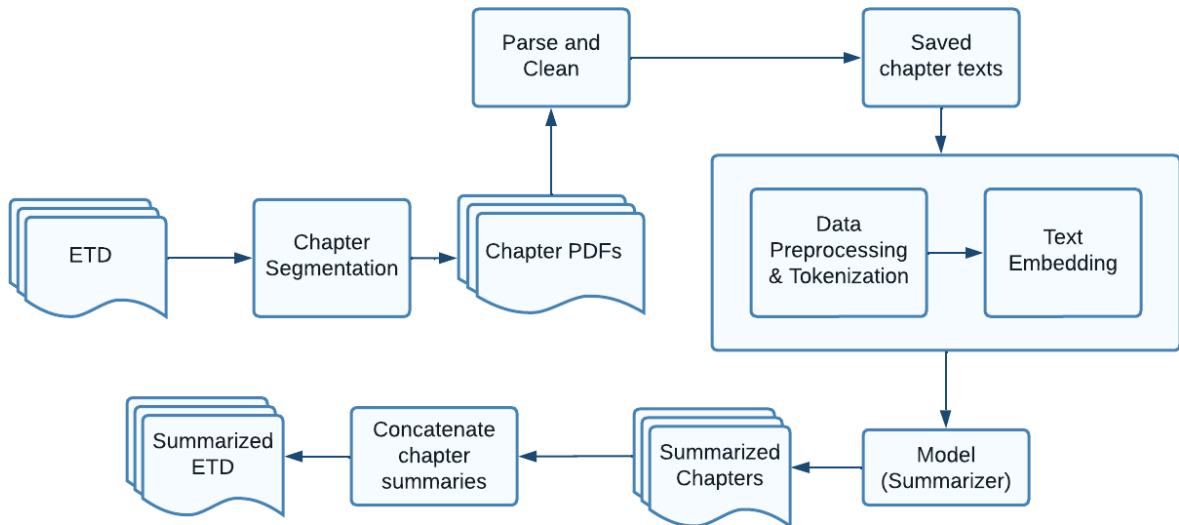


Figure 4.3: Summarization flow diagram

In the proposed pipeline (Figure 4.3), the ETD first passes through the ‘Chapter Segmentation’ module, which extracts every chapter from the ETD and stores them as chapter

## 4.4. SUMMARIZATION

PDFs. Each chapter is then passed through the parsing and cleaning service (which is an independent service) which parses the chapter PDFs, cleans the extracted text, and stores the chapters as text files.

The chapter text is then passed to the text summarizer model, which generates a summary of the complete chapter. Repeating the steps for every chapter input, we get a list of chapter summaries, which can either be concatenated to produce a summary of the entire ETD or utilized separately based on the end-user requirements.

As part of the project, although we planned to research new techniques for text summarization, the majority of the work focused on experimenting with various abstractive and extractive summarization algorithms, like TextRank [20], LexRank [21], Luhn’s Algorithm based summarizer [22], Latent Semantic Analysis based summarizer [23], and transformer-based models (like T5 [24], BART [25], etc.).

### 4.4.2 Methodology

The summarization pipeline is built as explained in the previous section. It has been containerized and is integrated into the workflow developed by Team 5. The current work is described as follows:

- Set the required parameters – model, summary length, etc.
- Pass a chapter text as input to the pipeline
- Summarize the chapter text using the model selected
- Generate a summary JSON (Figure 4.4)

```
{
  "chap_1.txt": [
    {
      "summary_text": "In this paper, Theobald discusses the importance of high-order arithmetic in solving large problems. He argues that it is necessary to be able to solve larger problems when the problem involves multiple orders of magnitude and/or multiple processes running simultaneously. For example, if a particle collides with an object at a fixed point in space , then the resulting product must be divided into smaller portions for each order of magnitude. This results in very large sums being needed to be solved. To illustrate how important these sums are, statistics are used to calculate the cross sections of products produced by different types of reactions. One example is the production or absorption of hydrogen using proton beams. By measuring the amount of energy released during the process, one can determine what part of the system needs to be shut down as soon as possible so that there will not be too much of a build-up"
    }
  ],
  "model": "pszemraj/bigbird-pegasus-large-K-booksum"
}
```

Figure 4.4: Chapter summary JSON

Initially, the summarization pipeline included steps like parsing and cleaning chapter PDFs. However, as part of the work completed earlier for Interim Report 3, this was separated into an independent cleaning module, hence only the path to the cleaned text needs to be passed to the pipeline. The generated JSON has two keys – summary and the algorithm used for summarization.

The summarization pipeline supports both abstractive (BigBird [11] pre-trained on the booksum dataset) and extractive summarization (TextRank [20], LexRank [21], and LSA [23]) models. However, it is to be noted that no model training is performed for summarization due to a lack of labeled training data. The pipeline only works in the inference mode.

In the current version of the IR system, summaries stored in the database are generated using TextRank [20] owing to its low processing time.

#### 4.4.2.1 Summarization using Extractive models

The extractive models supported in the pipeline are: TextRank [20], LexRank [21], and LSA [23]. As mentioned in the earlier section, these models do not generate new summaries, rather they pick the most important text(s) from the chapters as summaries. In the current workflow, the following are the primary steps.

- Read the chapter text file
- Perform word stemming
- Remove stop words
- Feed the processed text into the model
- Write the generated summaries into a JSON file

#### 4.4.2.2 Summarization using Abstractive models

The abstractive model currently being used (although this can be switched at any time before a run) is the Big Bird model [11] pre-trained on the booksum dataset [54]. The library used for implementing these pre-trained models is Hugging Face [28]. The advantage of this library is that it automatically performs tokenization and text embedding based on the transformer model selected. Hence, no manual pre-processing is performed while using Hugging Face transformer models, and the chapter text is simply fed into their model pipeline. Besides, the Big Bird [11] model, BART [25], and T5 [24] have also been tested and are supported by the pipeline.

### 4.4.3 Challenges

As mentioned in the above subsections, there is no training being done for summarization. Multiple models have been experimented with and integrated into the pipeline. Hence, for the purpose of selecting a final model for generating the summaries to be stored in the



## 4.5. CLASSIFICATION

database, model evaluation was planned. However, it was not carried out, owing to a few challenges as mentioned below.

Summaries generated by extractive and abstractive models vary based on their way of working. Abstractive models generate new text, whereas extractive models, on the other hand, pick out the most important lines/sentences from the text. Given that most of the ground-truth data consists of newly generated summaries, the evaluation results would be biased toward abstractive models. Hugging Face transformer models, particularly Big Bird, were used as abstractive models for the project. Big Bird [11] needs between five and ten minutes to compose a chapter summary. Therefore, the estimated time for 5000 ETDs (with an average of five to six chapters each) would be enormous.

The end-to-end workflow integration was the main focus due to time constraints. As a result, as the first version of the ETD summaries, TextRank [20] was used to build the summaries and store them in the database for further indexing.

### 4.4.4 Future work

As mentioned in the previous section, owing to time constraints, an extractive summarization model has been used currently to generate the summaries. However, such models do not always produce correct summaries, as the text might not contain any sentence(s) that can correctly signify the summary of the text, or if one exists, the algorithm might fail to extract it. Hence, abstractive models like transformer models are preferred. As the current summarization pipeline already supports transformer models, producing accurate summaries using these models can be an important future goal/task. Moreover, instead of directly using pre-trained transformer models, it would be useful if they can be further fine-tuned on some ETD-relevant dataset. Furthermore, the granularity can be varied, i.e., experiments at different granularity levels could be carried out as well, like at the section and paragraph levels.

## 4.5 Classification

### 4.5.1 Approach

In Figure 4.5, the steps for building classification models are highlighted.

Chapter classification is supported by our classification pipeline. The chapter classification's purpose is to produce discipline labels. Before we can build models, we must first create a training dataset, which is done by selecting a subset of the larger dataset based on the disciplines we are focusing on. Initially, chapters from the full PDF are segmented using the segmentation model. The dataset will then need to be preprocessed before models can be

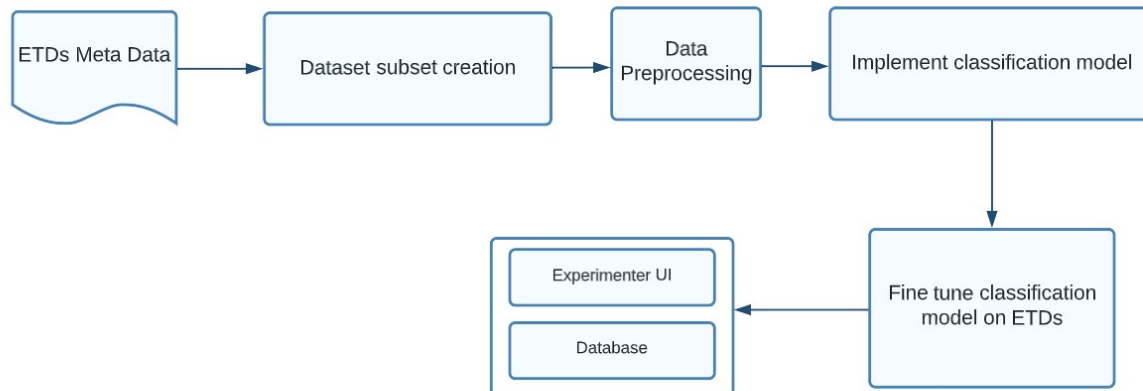


Figure 4.5: Classification flow diagram for chapter classification

trained on it. This step may include tokenization and label creation. For the classification models, we have employed Deep Transformer based models as well as a simple statistical model with fewer parameters. We have considered BERT, SciBERT, and Longformer for deep transformer-based models. For the statistical model, we have used a Support Vector Machine or Random Forest that is trained on word embeddings of text. In the following sections, we will first discuss the dataset creation and then describe the modeling approaches.

### 4.5.2 Chapter Classification: Methodology

One of our primary goals is to experiment with language models and traditional NLP methods for the task of classifying chapters into disciplines. The model will be fed segmented chapters in text format as input, and the output will be discipline labels. We assign at least two disciplines to each chapter because each chapter might relate to many disciplines. In contrast to the methods outlined in the metadata classification section, this problem presents some additional challenges due to the length of text in chapters. We have anticipated that, compared to abstracts, chapters will generally be significantly more verbose. Therefore, our approach must be capable of efficiently consuming and comprehending information from the input.

## 4.5. CLASSIFICATION

### 4.5.2.1 Data Collection and Preprocessing

We need the chapter-by-chapter split of text from the ETDs to classify the chapters. We have extracted the chapter texts from the ETDs in the segmentation ground truth dataset gathered by Javaid [29]. 132 chapter segmented ETDs from arXiv make up the segmentation ground truth dataset that Javaid gathered. We have gathered the chapter information for ETDs that are included in the ETD database as well as Javaid’s dataset. Overall, 40 ETDs in the ETD database were found in Javaid’s dataset. Only 30 of the 40 have metadata with discipline/department labels. So, we have compiled a dataset using chapters from 30 ETDs that have been segmented. Additionally, we have gathered data by running our segmentation pipeline on nearly 1500 ETDs. The segmentation pipeline yielded 4000 chapters in total.

After that, additional preprocessing was done to remove the tables from the text file that was created. Chapter-by-chapter PDF files were parsed and cleaned using the standalone service discussed in Section 4.3. However, for a few chapters, the parsed output was not ideal because of restrictions and flaws in the PDF Plumber library, which we presently use to remove graphics and tables from the chapter text. Occasionally, albeit not very often, the PDF Plumber failed to convert tokens from its character dictionary to ASCII values, leaving characters that are unintelligible. In the future, this error can be rectified by iterating over the output and converting the characters that weren’t converted to ASCII. After eliminating the subpar chapter texts, we had 3742 chapters left with which we fine-tuned our models.

The “Discipline” feature in the metadata was used to generate the classification labels for the chapters in the ETDs. It should also be noted that when the discipline labels were not available we used the department field as the label for the chapter. We observed that in many cases when both fields were available, they were quite similar to each other. Hence we used one as a substitute for the other. Due to the small dataset, we utilized a training set to test set ratio of 85% to 15%, to train and then evaluate the models that were trained. For the test set and training set, samples were randomly chosen from the full dataset. However, the stratified splitting preserved the distribution of labels in the test set. The chapter classification model trained can predict 27 different classes.

### 4.5.2.2 Chapter Classification using BERT and SciBERT

BERT is a transformer-based language model that has been trained on a huge general English text corpus with a strong understanding of language. We believe that it can be appropriately applied here for the use case of chapter classification. We have developed a pipeline for classifying chapter texts based on disciplines. The chapter texts in the dataset amassed by our team were preprocessed to eliminate tables and figures by our clean and parse service. While storing the results of 5000 ETDs in the database, we have used the cleaned text already present in the database rather than reading and preprocessing the PDF file each time the classification pipeline is executed. Since BERT has a sequence limit of 512, we have selected

and classified the first 512 words in the chapter. We felt this was not problematic because the average length of the chapter text was 255 tokens (in the dataset used for training).

We have also used the summaries generated by our team to fine-tune BERT on the summaries rather than the entire text. We didn't notice any significant improvement in the performance of models tuned on summaries over the entire text as the mean length of the entire text was well within the maximum sequence limit of BERT. We have developed a pipeline to train the model and gather the results after training.

Since the results presented in a SciBERT paper [32] are very encouraging for a variety of disciplines, we have also decided to use the SciBERT language model. SciBERT, which is used to understand scientific texts, is actually quite similar to BERT but has the additional advantage of being trained in scientific corpora. A pipeline for training and evaluating SciBERT has also been created. Due to the SciBERT pretraining on scientific text, we observe that it is able to outperform BERT on our evaluation set. The model architecture used for this task is the same as the model design described in Section 4.5.4.2, with the input changed to chapter text.

### 4.5.2.3 Chapter Classification using Longformer

Longformer is an alternative larger transformer model, which has been trained on a language modeling task similar to BERT and SciBERT but with a key difference in architecture. It uses a different attention mechanism allowing it to increase the sequence limit from 512 to 4096. Now with this higher sequence limit, we do not need to truncate the chapters to 512 during preprocessing. We can assume that the first 4096 tokens within the chapter would provide enough information to accurately classify the entire chapter. In terms of architecture, we apply a similar architecture to simple BERT-based classification, but now we use Longformer to provide contextual representations. The architecture is illustrated in Figure 4.6. We have developed pipelines for this method to train the model. The model architecture used for this task is the same as the model design described in Section 4.5.4.2 with the model changed to Longformer.

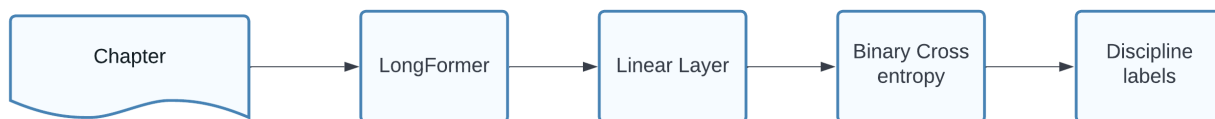


Figure 4.6: Chapter Classification using Longformer

## 4.5. CLASSIFICATION

### 4.5.2.4 Chapter Classification based on chapter summary

Before building the models, we hypothesized that the majority of chapter texts in our dataset would have more tokens than 512. This is problematic because, as previously indicated, most language models have a 512 tokens restriction. Using larger models, such as Longformer, is one method to remedy this. However, these models are inefficient and need a lot of memory to execute. A more resource-efficient approach would involve creating chapter summaries first, then classifying those summaries. This way, the summaries would condense all the material into 512 words. Once the summaries are generated we can use a BERT or SciBERT based classification pipeline that has been described above. Since we have generated chapter summaries, it makes sense to follow this approach. A pipeline for this method has been created. This approach has been illustrated in Figure 4.7.

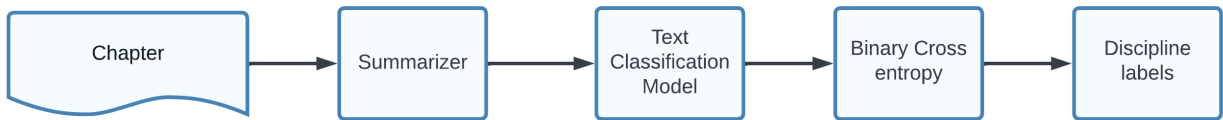


Figure 4.7: Chapter Classification based on summaries

### 4.5.2.5 Chapter Classification using SVM and Random Forest

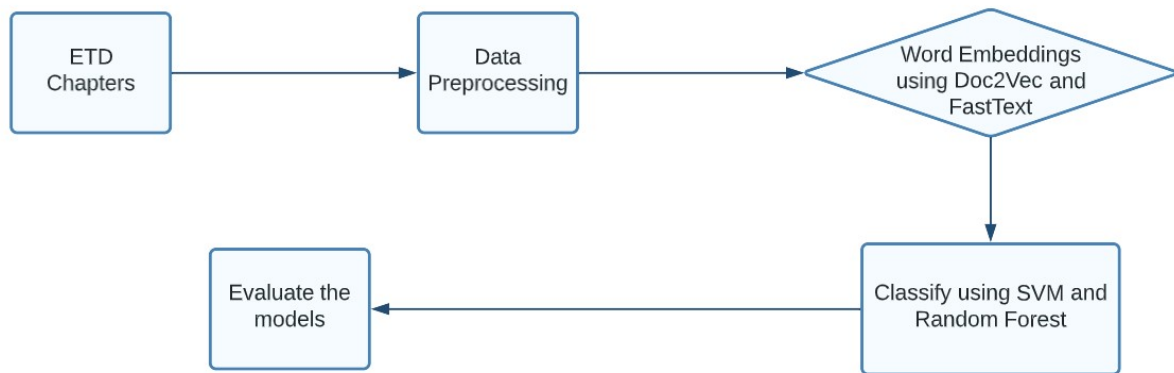


Figure 4.8: Classification using SVM and Random Forest

In Figure 4.8, we highlight the major steps followed during the replication of Palakh's thesis [26] to set up a baseline for other models.

First, the data was preprocessed. General preprocessing steps performed during NLP tasks such as stop word removal, punctuation removal, lemmatization, and conversion of all the text into the lower case were completed, and the results stored.

Word embeddings of the full text were generated using two models, namely FastText and Doc2Vec. The word embeddings are not ideal for long documents such as ETDs as the order of words is not considered, resulting in the loss of important information. But for baseline implementation, word embeddings were generated.

Since we want multiple discipline labels for each chapter, as each chapter in a thesis may represent multiple disciplines, we considered the classification task at hand as a multi-label classification problem. Most of the classifiers implemented in Scikit-learn are built for single-label classification. Thus to solve the multi-label classification problem, the multi-label classification problem can be transformed into multiple single-label classification problems so that we can use the pre-existing algorithms to solve it. As was done by Palakh, we have considered the LabelPowerset approach for the dataset. We have finally developed a pipeline to train SVM (Support Vector Machine) and Random Forest to set us up with a baseline.

### 4.5.3 Model Evaluation

We have been able to train the BERT and SciBERT models described above on our current ETD chapter classification dataset, and a dataset of chapter summaries. The summaries contain the condensed versions of the documents and are useful in cases when the number of tokens exceeds the sequence limit of 512 tokens. All our data pipelines and training pipelines were developed in PyTorch. For the Language Models, we used Hugging Face’s Transformers library. Using this library, we were able to obtain the pre-trained weights for BERT, SciBERT, and Longformer. All of our models have been trained on the following hyperparameters:

- Optimizer: AdamW
- Learning Rate: 5e-5
- Batch Size: 8
- Epochs: 8

Once the training was complete, we performed and evaluated both the summary and chapter models on a small evaluation test set consisting of 749 chapters. The performance metrics of Accuracy and F1 have been recorded and are mentioned in Table 4.2.

## 4.5. CLASSIFICATION

Table 4.2: Evaluation of classification models

Model Name	Accuracy (%)	F-1 Score (%)	Chapter/Summary
BERT-base	64.75	56.28	Chapter
SciBERT	80.77	77.5	Chapter
BERT-base	63.01	54.41	Summary
SciBERT	70.89	67	Summary

When we look at the results of the chapter and summary models we can see that the chapter models perform better than the summary models, across the board. For both of the models, i.e., SciBERT and BERT, their summary-trained counterparts are worse than the chapter models. This result was unexpected, because generally one would assume the chapters to be so long that once we use transformer-based models on this dataset, information would be truncated. In practice, however, that is not the case. We see that the chapter texts are generally within 512 tokens for most documents and the average number of tokens in the chapters is around 300. This suggests that when we are generating summaries from the chapter text we are effectively reducing the amount of information available. Due to these reasons, it can be understood that the summary models are worse on the test set when compared to the chapter models. From the results, it can also be noted that the chapter models are quite good at classifying the chapters. In our qualitative experiments we tested the code with some new documents and the output of the models was satisfactory.

### 4.5.4 Inference on ETD subset

As is clearly shown in Section 4.5.3, SciBERT fine-tuned on chapter text performed the best of all. The performance of the models fine-tuned on chapter summaries wasn't as impactful as we hypothesized, because the average length of the chapter text was less than the maximum permissible token limits of BERT. Classification performance not only depended on the performance of the segmentation model but also on the model generating summary instead of just segmentation models. Thus we have used the SciBERT model fine-tuned on the chapter text to store the classification labels on 5000 ETDs subset.

### 4.5.5 Future Work

#### 4.5.5.1 Metadata Classification

##### 4.5.5.1.1 Methodology

Library metadata is a very important source describing a crucial piece of information through which information can be stored and retrieved easily. Metadata ensures that the data can

be found, accessed, and interpreted easily. The absence of information in metadata can be considered a barrier to information retrieval. While combing through the metadata of 500k ETDs, we found out that a lot of fields in the metadata are missing. According to one of our prior studies, 65,955 fields of year data, 232,653 fields of department data, and 166,690 fields of discipline data were missing out of 450k ETDs collected by Uddin et al. [3]. Since the absence of metadata is very detrimental to efficient information storage and retrieval, we were planning on developing traditional NLP methods and deep transformer-based methods to classify the discipline given an abstract. We are posing this problem as a multi-label classification problem where each abstract may be assigned multiple disciplines.

#### 4.5.5.1.2 BERT-based Metadata Classification

We have been able to develop a BERT-based pipeline for classifying the metadata into disciplines, since the input to the models, i.e., the abstracts, are generally short texts. We do not need to worry about information loss due to the text length limit of BERT models. This pipeline is able to consume a dataset of ETD abstracts, convert them into batches, and perform necessary preprocessing which for transformers includes tokenization. Tokenization breaks the sentence down into smaller tokens which may be words or sub-words. After this, we split the dataset into training and validation so the model can be thoroughly evaluated. We can explain the model design used for this task. First, the text is tokenized using a BERT tokenizer. This step is equivalent to converting each compound word in the text to a number that the model can interpret. Within this step, we also truncate the text content to 512 words. When this tokenized text is provided as input, the BERT model processes the text and generates a 768-dimensional representation of the entire sentence. We take this vector and pass it through a linear layer that returns logits that have the same length as the number of classes. On these logits, we apply the BinaryCrossEntropy loss which can be used in multi-label classification. This loss essentially treats each value in the logits as a logistic regression problem, i.e., 1 if the class is present, otherwise 0. Once the model is trained, we can round the logits and get the class predictions. The architecture is illustrated in Figure 4.9.

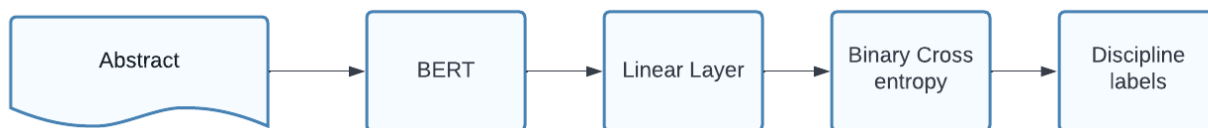


Figure 4.9: Metadata classification using BERT

The input to chapter classification (when applied to chapter text) and metadata classification (when applied to the abstract) are very similar, so we can also use the chapter classification model to classify the missing disciplines in metadata.



## 4.5. CLASSIFICATION

### 4.5.5.2 Knowledge Distillation

Knowledge distillation is the process through which a smaller network is trained to leverage the knowledge gathered by large networks. Through knowledge distillation, we are effectively training a smaller network to replicate the performance of the larger network. This is highly advantageous because fine-tuning and getting inferences from a large model is computationally very expensive and time-consuming. Since the distilled models are lighter than the original models, further fine-tuning can be done quite easily. In the future, the inference time of our classification system can be further improved by distilling SciBERT into a smaller model.

### 4.5.5.3 Citation Intent Classification

Citing research papers referred to is an integral part of scientific works such as ETDs. The authors are required to cite the work done by others previously because the idea used is not from them. Citations give your work more credibility, advise readers of where to get a more in-depth explanation of it that is pertinent to the content, and acknowledge the person whose work or concept you are employing.

For the analysis of scientific works and the relationships between them, determining the citation’s intent is essential. Finding the purpose of a citation helps to determine the publication’s influence and can also point interested readers in the direction of the kinds of additional readings they might find interesting.

The development of a Citation Intent Classification dataset from ETDs is the initial stage in the classification of citation intent. The purpose of citations can be marked in a manner that is analogous to the SciCite dataset [1]. This allows citations to be classified into the following three classification labels: Background, Method, and Result Comparison. To assure the quality of the dataset, at least two separate individuals can verify the prepared dataset. The generated dataset can be used to evaluate how well the trained and tuned models function.

The approach’s second stage would involve investigating various deep learning or language models to classify the citation intent. After classifying the citation intent using the model that performed the best out of all those that would be trained, prompt tuning [56] can be investigated to effectively fine-tune the model. Knowledge Distillation [7] of the best larger model trained to a smaller model can also be attempted to decrease the training duration and also to fine-tune the learning of the larger model to the domain-specific datasets.

### 4.5.5.4 Investigating Chapter Lengths

The average chapter length of the training set was less than 512 tokens. This runs counter to our assumption that chapter text would typically be longer than 512 tokens. We believe

that improvements to the clean and parse pipeline will resolve this issue. Currently, the clean and parse service is not able to extract all the chapter tokens from the chapter PDF. However, this matter may be looked upon in the future.

## 4.6 User Interface

The Figure 4.10 experimenter portion of the front-end accepts a new PDF as input. First, the uploaded PDF is added as a new ETD using Team 1’s add new ETD API. The API returns the newly added ETD’s ID as a response. Next, Team 5’s workflow is triggered. The workflow is designed in such a way that it calls the services in the following order:

1. Get ETD PDF using ETD ID
2. Segmentation service
3. Clean and parse service
4. Summarization service
5. Classification service
6. Save results to the database

The status of the workflow can be checked using the “get workflow status” API. Once the workflow execution is complete, the generated results are fetched from the database and displayed in the front-end.

## 4.6. USER INTERFACE

# Team 4

## Title : Title of the ETD

[Link to meta data](#)

Choose Summarization Model

Model ▼

Choose Classification Model

Model ▼

GO

S. No.	Segmented PDF	Chapter Summary	Classification Label
1	<a href="#">Chapter1.pdf</a>	This is the summary of chapter 1	Physics
2	<a href="#">Chapter2.pdf</a>	This is the summary of chapter 2	Electromagnetics
3	<a href="#">Chapter3.pdf</a>	This is the summary of chapter 3	Quantum
4	<a href="#">Chapter4.pdf</a>	This is the summary of chapter 4	Physics

Figure 4.10: Experimenter User Interface wire-frame

## 4.7 Tools

To successfully complete the project we worked with the following tools and technologies.

**Python:** Our primary language for development is Python as it allows for fast prototyping when researching different solutions. Furthermore, Python offers the best support for various kinds of NLP and Deep Learning/Machine Learning libraries overall, making it an ideal choice.

**Docker:** Docker [37] provides a completely focused environment that is catered towards running your specific solution in any kind of environment. It contains all the necessary prerequisites libraries and system-level binaries required for functioning. For our project, we build containers for each of our services, which are summarization, segmentation, and classification.

**Python utility libraries:** We use the PyPDF2 [51] and PDF Plumber [40] libraries to parse the PDFs. Through these libraries, we are extracting text information that is used for the classification and summarization pipelines. We also use the Requests library to fetch and send data through APIs [43].

**GitLab:** GitLab will host the code for the team and allows for easy maintenance of various feature branches. Furthermore, through GitLab, CI/CD pipelines can be set up for deployment [38].

**PyTorch:** We are working with PyTorch [27] which is a machine learning library providing methods to build layers, models, and data pipelines for this project. PyTorch will accelerate development time and also allows for hardware-accelerated training.

**Hugging Face:** Hugging Face [28] is a library built around PyTorch that provides support for state-of-the-art Transformer models. Hugging Face provides easy access to pre-trained weights for transformer architectures like BERT, etc., for faster experimentation.

**NLP libraries:** We use libraries like spaCy [49] and NLTK [50] which provide APIs to perform some common NLP tasks like lemmatization, removing stop words, etc. We are also using sumy [39] for performing text summarization. We rely on other libraries like NumPy [48] for working with arrays and Pandas [46] for working with dataframes.

# Chapter 5

## Implementation

### 5.1 Tasks and Timeline

Tables 5.1, 5.2, and 5.3 detail the activities completed and the completion timeline for segmentation, summarization, and classification, respectively.

#### 5.1.1 Segmentation Tasks and Timeline

Tasks Assigned: Dharneeshkar

Table 5.1: Tasks and Timeline for Segmentation

Task	Due Date	Remarks
Literature Review	IR2	Complete
Setting up Javaid's pipeline	IR3	Complete
Setting up segmentation code in Team 1's container	Final Report	Complete
Storing segmentation results for 5000 ETDs to the database	Final Report	Complete
Collaborating with Team 5 to automate our service	Final Report	Complete
Integrating with the User Interface	Final Report	Complete

### 5.1.2 Summarization Tasks and Timeline

Tasks Assigned: Abhilash and Deepak

Table 5.2: Tasks and Timeline for Summarization

Task	Due Date	Remarks
A Basic Summarization pipeline	IR2	Complete
Data Preprocessing/Cleaning	IR3	Complete
Implementing at least 3 different models	IR3	Complete
Qualitative evaluation and experimentation for final model selection	Final Report	Complete
Storing summaries of 5000 ETDs in the database	Final Report	Complete
Collaborating with Team 5 to automate our service	Final Report	Complete
Integrating with User Interface	Final Report	Complete

### 5.1.3 Classification Tasks and Timeline

Tasks Assigned: Kaushik, Deval, and Aditya

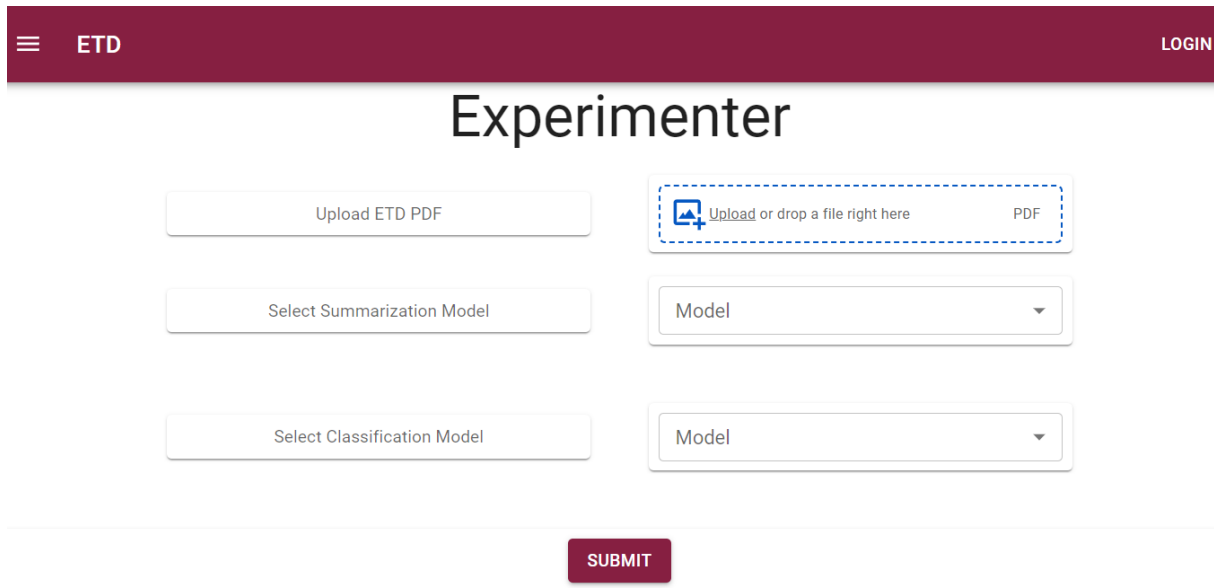
Table 5.3: Tasks and Timeline for Classification

Task	Due Date	Remarks
Establishing a baseline by classifying disciplines in the full text of PQDT dataset	IR2	Complete
Establishing a baseline for metadata classification by classifying the disciplines in the 500k metadata dataset.	IR2	Complete
Creating a labelled dataset for classification of disciplines from ETD repository	IR3	Complete
Training chapter classification models including BERT, SciBERT and Longformer	IR3	Complete
Fine-tuning the models for best performance	Final Report	Complete
Storing the chapter classification results of 5000 ETDs in the database	Final Report	Complete
Collaborating with Team 5 to automate our service	Final Report	Complete
Integrating with User Interface	Final Report	Complete

# Chapter 6

## User Manual

### 6.1 Experimenter Page



The screenshot shows the 'Experimenter' web page. At the top is a dark red header bar with a hamburger menu icon, the text 'ETD', and a 'LOGIN' link on the right. Below the header, the word 'Experimenter' is centered in a large, black, sans-serif font. The main content area contains several interactive elements: a button labeled 'Upload ETD PDF', a dashed blue box with a file upload icon and the text 'Upload or drop a file right here' followed by 'PDF', a button labeled 'Select Summarization Model', a dropdown menu labeled 'Model', a button labeled 'Select Classification Model', and another dropdown menu labeled 'Model'. At the bottom center is a dark red 'SUBMIT' button.

Figure 6.1: Experimenter User Interface web page

Figure 6.1 shows the page where users can experiment with different summarization and classification models. React [55] is used for front-end development. Initially, to access this page the user needs a login. To experiment with a new ETD, the user can upload an ETD PDF file. The uploaded ETD is saved in the repository using an API call. When the user clicks the submit button, Team 5's workflow is triggered using an API call. The workflow runs all the required services and saves the results to the database and file system.

The generated results are displayed in Figure 6.2. The summaries for each chapter and the top two classification labels are presented to the user.

Segmented PDF	Classification Label	Chapter Summary
Chapter 1	Computer Science, Electrical and Computer Engineering	Residual networks (ResNets) have recently achieved state-of-the-art on challenging computer vision tasks. We introduce Resnet in Resnet (RiR): a deep dualstream architecture that generalizes ResNets and standard CNNs and is easily implemented with no computational overhead. RiR consistently improves performance over ResNets, outperforms architectures with similar amounts of augmentation on CIFAR-10, and establishes a new state-of-the-art on CIFAR-100.
Chapter 2	Computer Science, Statistics	The trend towards increasingly deep neural networks has been driven by a general observation that increasing depth increases the performance of a network. Recently, however, evidence has been amassing that simply increasing depth may not be the best way to increase performance, particularly given other limitations. Investigations into deep residual networks have also suggested that they may not in fact be operating as a single deep network, but rather as an ensemble of many relatively shallow networks.
Chapter 3	Computer Science, Statistics	As a result, we are able to derive a new, shallower, architecture of residual networks which significantly outperforms much deeper models such as ResNet-200 on the ImageNet classification dataset. We also show that this performance is transferable to other problem domains by developing a semantic segmentation approach which outperforms the state-of-the-art by a remarkable margin on datasets including PASCAL VOC, PASCAL Context, and Cityscapes.
<div> <div>Rows per page: 100 ▾</div> <div>1–3 of 3</div> <div>&lt; &gt;</div> </div>		

Figure 6.2: UI displaying segmented chapters and summaries

The models for summarization and classification can be selected by the user using the drop-down box shown in Figures 6.3 and 6.4, respectively.

Select Summarization Model

Select Classification Model

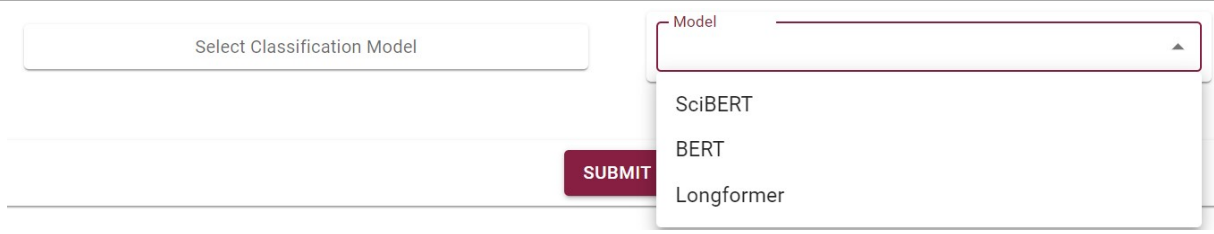
Model ▾

TextRank  
BART  
BigBird

Figure 6.3: Dropdown box to select summarization model



## 6.1. EXPERIMENTER PAGE



The image shows a web interface for selecting a classification model. On the left, there is a text input field with the placeholder text "Select Classification Model". To the right of this field is a dropdown menu. The dropdown menu is currently open, showing a list of model names: "SciBERT", "BERT", and "Longformer". Above the list, the word "Model" is visible, indicating the selected category. Below the dropdown menu, there is a red button with the text "SUBMIT" in white capital letters.

Figure 6.4: Dropdown box to select classification model

However, since the back-end of the experimenter page is running the workflow automation done by Team 5, it currently supports only default models for summarization and classification. The default models supported are TextRank for summarization and SciBERT for classification. The default models for summarization and classification were selected by analyzing the trade-offs based on the time taken for the model to run and the performance of the models.

We have attached two videos, **team4\_experimenter\_demo.mp4** and **team4\_documentView\_demo.mp4**. The description of each is as shown in Tables [6.1](#) and [6.2](#), respectively.

Table 6.1: Description of video team4\_experimenter\_demo.mp4

Timestamp	Description
0:00	The home page of the UI.
0:05	Navigating to the Experimenter page.
0:06	Uploading ETD PDF, and selecting summarization and classification models.
0:20	Receiving alerts indicating the successful addition of an ETD into the file system, as well as that the workflow has successfully been generated and is now running.
0:24	Switching over to the back-end to show Team 5's workflow.
0:25	We can observe Team 5's 4 APIs here as indicated: Generate Grammar- Generates context-free grammar representation. Generate workflow- Creates workflow for our services. Run workflow- Starts a workflow with ETD ID. Get workflow status- Returns the current status of the workflow.
0:27	A key is inputted to check the workflow status; we can observe that the process is still running.
0:39	Switching back to the front-end.
0:43	We can observe the output in the form of the chapter title, classification labels, and chapter summary.
0:49	Receiving an alert indicating the completion of the workflow.

Table 6.2: Description of video team4\_documentView\_demo.mp4

Timestamp	Description
0:00	The home page of the UI.
0:03	Searching for a topic.
0:06	Since our ETD subset is from UCB, we have filtered the search result by UCB.
0:11	Selecting an ETD.
0:15	We can observe the document view page output designed by Team 2.
0:18	We can observe the results generated by Team 4, in the form of the chapter title, classification labels and chapter summary. It should be noted that this type of output is only generated for the first 5000 ETDs.
0:34	We navigate back to the Home page and select an ETD with an ID of more than 5000 to show the different generated output.
0:43	We can observe that no table is shown for this ETD since the results for ETDs having an ID of more than 5000 are not stored.

## 6.2 Future Work

We have currently used Team 5’s workflow automation as the back-end for our experimenter page. As explained, when a new ETD is uploaded, the workflow automation spins up the required containers sequentially based on the Docker images given by Team 4 and stores the results in the file system and database. The user must wait for a short period of time until the workflow automation is completed and the user interface is populated. This latency can be reduced in the future. Due to a lack of time, we haven’t integrated other models into the workflow automation even though the Docker images compiled by us are capable of running multiple models for summarization and classification. We have chosen a default model for summarization and classification by considering trade-offs between the time taken by the individual model and the performance of the model. The additional integration of models other than default models can be taken up in the future.

# Chapter 7

## Developer Manual

In this section, we describe how to set up, develop, and contribute to our existing services and pipelines. We cover the installation setup, and then for each of the services, we explain how to run the code and train models. The API endpoints are listed for each of the services.

### 7.1 Installation Setup

We have already performed this setup within our containers but for the sake of completeness we describe the setup. Most of our code relies on GPU acceleration, so we would suggest gaining access to a GPU-accelerated container. Then to enable acceleration we would suggest installing Cuda Toolkit and CuDNN. Once that is done, we can check if it is working as expected or not by running “nvidia-smi”. On a system level we would like to ensure the following libraries and binaries are installed.

- Cuda Toolkit[\[42\]](#)
- CuDNN[\[42\]](#)
- Nvidia Drivers
- Python 3.8[\[41\]](#)

Next install the necessary libraries which are the dependencies for each of our services. This includes the following:

1. Hugging Face [\[28\]](#)
2. PyTorch [\[27\]](#)
3. Tensorflow-gpu [\[44\]](#)
4. scikit-learn [\[45\]](#)
5. Pandas [\[46\]](#)

## 7.2. APIs

6. Matplotlib [47]
7. Numpy [48]
8. spaCy [49]
9. NLTK [50]
10. PyPDF [51]
11. PDF Plumber [40]
12. sumy [39]

## 7.2 APIs

We have used the following APIs to access the objects from the database and file system and store the objects produced by us (summary and classification labels) in the database.

### Summarization

1. Getting Object IDs and Object metadata:  
`https://team-1-flask.discovery.cs.vt.edu/v1/etds/<etd-id>/objects?type=cleaned__text`
2. Get the object: `https://team-1-flask.discovery.cs.vt.edu/v1/objects/<object-id>/file`
3. Uploading summary to the database: `https://team-1-flask.discovery.cs.vt.edu/v1/objects/<object-id>/summarisation`

### Classification

1. Getting Object IDs and Object metadata:  
`https://team-1-flask.discovery.cs.vt.edu/v1/etds/<etd-id>/objects ?type=cleaned__text`
2. Get the object: `https://team-1-flask.discovery.cs.vt.edu/v1/objects/<object-id>/file`
3. Uploading classification results to the Database: `https://team-1-flask.discovery.cs.vt.edu/v1/ objects/<object-id>/classification`

## 7.3 Instructions

### 7.3.1 Classification

To run the classification code to store classification results in the database, navigate to and clone <https://code.vt.edu/ano22/cs5604-team4-classification> GitLab repository and run the Final\_Classification\_Store\_1.ipynb jupyter notebook.

To run our classification model pipelines and training, access the Team 4 folder on cloud.cs and navigate to data/Team4/Classification. Then run the following files with GPU support:

- To train the BERT model, run the Deep\_classifier.ipynb.
- To train Longformer model, run the Deep\_classifier\_longformer.ipynb.
- To train SciBERT, run the BERT\_based\_chapter\_classifier\_SciBERT.ipynb.
- To gather inferences of BERT and SciBERT models, run test\_inference.ipynb
- While running test\_inference.ipynb, give the correct chapter path, choose the right model type – either bert-base-uncased or allenai/scibert\_scivocab\_uncased (SciBERT) – and choose the right model weights file:
  - /mnt/data/Team4/Classification/saved\_models/new\_chapter\_bert\_5.pt – for BERT-base trained on chapter text
  - /mnt/data/Team4/Classification/saved\_models/chapter\_scibert\_8\_final\_weights\_chapter.pt – for SciBERT trained on chapter text
  - /mnt/data/Team4/Classification/saved\_models/new\_Summary\_bert\_5.pt – for BERT trained on summaries
  - /mnt/data/Team4/Classification/saved\_models/new\_summary\_scibert\_8.pt – for SciBERT trained on summaries

We have also containerized the classification service which can be pulled by a developer to run the models and code in a pre-defined environment. The name of our image is **dexuiz/-classification:0.0.3**. This image contains our latest code and models for classification. We can now follow the steps below to get a running model.

- First we need to pull the image from the Docker repository.
  - `docker pull dexuiz/classification:0.0.3`

### 7.3. INSTRUCTIONS

- Now we want to run this image. The command below runs the container and starts an interactive container with bash access. If you wish to attach a volume to this container you can use the flag `-v`, specifically add the following to the command below `-v PATH_TO_FOLDER:/data`. This will mount the folder specified in the local machine to the folder `/data` in the Docker container.

```
- docker run --rm -it dexuiz/classification bash
```

- Once the above command successfully executes we will have a bash command line within the Docker container. Now we can run the classification script within the container. In the command below for the flag `--input-path` you need to provide the path to the input file which can be either a cleaned text file or a PDF file. The default location for the output is `/out` but if you need to change that folder you can provide that using the `--output-path` flag.

```
- python3 classifier.py --input-path ./data/chap.txt --output-path  
  ./out
```

- Now the script will compute the labels for this file and place the output JSON in the provided output folder.

#### 7.3.2 Summarization

To run the summarization code directly on your machine, navigate to and clone <https://code.vt.edu/harishbabu/cs5604-f22-team-4/-/tree/Summarization> GitLab repository. Running instructions are:

- Add required parameter values (model, summary max length, etc.) to the `internal_config.json` (Figure 7.1). All the required parameters and hyper-parameter values need to be specified in this JSON file.
- Run » `python summarization.py --path <path_to_chapter_text> --path : mandatory argument`

```
{
  "summarizer_model": "pszemraj/bigbird-pegasus-large-K-booksum",
  "transformer": true,
  "summarizer_tokenizer": "",
  "summarizer_framework": "pt",
  "summarizer_min_length": 16,
  "summarizer_max_length": 512,
  "do_sample": false,
  "sub_path": "chapters",
  "max_seq_len": 4096,
  "no_repeat_ngram_size": 3,
  "encoder_no_repeat_ngram_size": 3,
  "repetition_penalty": 3.5,
  "num_beams": 4,
  "early_stopping": true,

  "model_name_json_key": "model",
  "max_summary_sentences_sumy": 20,

  "cascade": true
}
```

Figure 7.1: Summarization Internal Config

Similar to the Classification service we have also containerized the summarization service in DockerHub. In the following section we have included instructions to quickly set it up and run it.

- First we need to pull the image from the Docker repository.
  - `docker pull dharneeshkar/summarization:0.9`
- Now we want to run this image; we can do that using the command provided below. If you wish to attach a volume to this container you can use the flag `-v`, specifically add the following to the command below `-v PATH_TO_FOLDER:/data`. This will mount the folder specified in the local machine to the folder `/data` in the container.
  - `docker run --rm -it dharneeshkar/summarization:0.9 bash`
- Once the above command successfully executes we will have a bash command line within the Docker container. Now we can run the classification script within the



### 7.3. INSTRUCTIONS

container. In the command below for the flag `--input-path` you need to provide the path to the input file which has to be a text file containing cleaned text. The default location for the output is `/out` but if you need to change that folder you can provide that using the `-output-path` flag.

```
– python summarization.py --input-path ./data/ --output-path ./out
```

- Now the script will generate a summary for this file and place the output JSON file in the provided output folder.

#### 7.3.3 Summarization API Driver

The summarization driver script can be found here: <https://code.vt.edu/harishbabu/cs5604-f22-team-4/-/tree/Summarization/driver.py>. The driver script encapsulates all of the API calls to run the summarization pipeline: fetch cleaned chapter text, summarize, and save in the database. The working of the driver can be summarized as follows:

1. Read ETD IDs from text file containing ETD IDs
2. For each ETD ID get the list of object IDs using the `get_obj_by_etd_id` API
3. For each object ID fetch the object (in this case, chapter text) using `get_obj_file_by_obj_id` API
4. For each object/file fetched, perform summarization and save the summary JSON into the database using the `save_summary` API

To run the driver script, use:

```
» python driver.py
```

**Note:** Keep the `etd_id.txt` file in the same directory as the driver script

#### 7.3.4 Segmentation

We developed a segmentation service to split an entire ETD into chapters. The code for the service is also available at <https://code.vt.edu/ano22/cs5604-team-4-segmentation>. You can run it following the instructions specified in the readme file.

Apart from using the code directly, you can also use it as a container as we have containerized the segmentation service. In the following section, we have included instructions to quickly set it up and run it.

- First we need to pull the image from the Docker repository.

```
– docker pull dharneeshkar/segmentation:0.4
```

- Now we want to run this Docker image. We can do that using the command provided below. If you wish to attach a volume to this container you can use the flag `-v`, specifically add the following to the command below `-v PATH_TO_FOLDER:/data`. This will mount the folder specified in the local machine to the folder `/data` in the container.

```
– docker run --rm -it dharneeshkar/segmentation:0.4 bash
```

- Once the above command successfully executes we will have a bash command line within the Docker container. Now we can run the classification script within the container. In the command below for the flag `--input-path` you need to provide the path to the input file which has to be a PDF file containing the entire ETD. The default location for the output is `/out` but if you need to change that folder you can provide that using the `--output-path` flag.

```
– python segmentation.py --input-path ./data/ --output-path ./out
```

- Now the script will read the entire ETD and segment it into smaller chapters. Once that is done it will write each chapter back as PDFs in the output folder. So, after completion, the output folder will be populated with chapter PDFs.

### 7.3.5 Parsing and Cleaning

The Summarization and Classification services require cleaned text input to properly generate their respective outputs, so to prevent the repetition of code we decided to create a parse and clean service. This service reads a chapter PDF after it has been segmented and then reads all the text present in it after filtering all the tables and images. The final output of this service is a chapter-cleaned text that can be used in downstream tasks.

To run the parsing and cleaning code directly on your machine, navigate to and clone <https://code.vt.edu/harishbabu/cs5604-f22-team-4/-/tree/Summarization> GitLab repository.

Apart from running the code directly, we have also deployed it as a container similar to other services so you can directly pull that and start using the parse and clean service directly. The following instructions can be followed to do that.

- First we need to pull the image from the Docker repository.

```
– docker pull dharneeshkar/parse_and_clean:0.1
```

- Now we want to run this image. We can do that using the command provided below. If you wish to attach a volume to this container you can use the flag `-v`, specifically

## 7.4. WORKFLOW AUTOMATION APIS

add the following to the command below `-v PATH_TO_FOLDER:/data`. This will mount the folder specified in the local machine to the folder `/data` in the container.

```
– docker run --rm -it dharneeshkar/parse_and_clean:0.1 bash
```

- Once the above command successfully executes we will have a bash command line within the Docker container. Now we can run the classification script within the container. In the command below for the flag `--input-path` you need to provide the path to a folder that contains all the PDF file chapters of the ETD that have to be cleaned. The default location for the output is `/out` but if you need to change that folder you can provide that using the `--output-path` flag.

```
– python parse_and_clean.py --input-path ./data/chapters  
  --output-path ./out
```

- Now the script will read the folder, which will have a set of chapters in it, and then create the cleaned text versions of these chapters, and write them to the output folder.

## 7.4 Workflow Automation APIs

In collaboration with Team 5, we performed workflow automation to generate a flow of processes that will be completed automatically once we obtain a new ETD. This workflow is controlled by a few APIs which have been described below:

- **Generate Workflow:** First, you need to run Generate Workflow to create a workflow for the services. Input to the API is the ETD ID of the file that has to be processed. Along with this, in the body of the request, you need to add the environment of the services. An example of the body and the URL has been included below. Once you run this script with the correct environment file you will get a workflow ID in return and that can be used in the next stage.

```
– URL: https://reasoner.discovery.cs.vt.edu/generateWorkflow/:workflowId
```

```
– Body:
```

```
1 '{  
2   "env": [  
3     {  
4       "service_id": "all",  
5       "service_env": [  
6         "ETDID={{etd_id}}"  
7       ]  
8     }  
9   ]  
10 }
```

```

9 |      ]
10|    },

```

- **Run Workflow:** After running the previous command you would obtain a workflow ID and you can use that to start the workflow. The command for that has been provided below. Once this command completes you will obtain a workflow key and you can use that to run the API provided below to look at the status of the request.

– **URL:** <https://team2020-airflow.discovery.cs.vt.edu/workflow/run>

– **Body:**

```

1 {
2   "args":["{{ workflowId }}","2020-10-10"],
3   "service_names":[
4     "curl-get-etd", "segmentation", "curl-save-chapters"
5   ]
6 }

```

- **Get Workflow status:** To monitor the status of the workflow and see how far along is it, you can use the get workflow status API. The API URL requires a workflow key that would have been generated by the previous API. This API will return the logs of the workflow that can be used for monitoring.

– **URL:** <http://team2020-airflow.discovery.cs.vt.edu/workflow/run?key=key>

# Bibliography

- [1] Cohan, Arman, Waleed Ammar, Madeleine van Zuylen, Field Cady. “Structural Scaffolds for Citation Intent Classification in Scientific Publications.” Proceedings of the 2019 Conference of the North, Association for Computational Linguistics, 2019, pp. 3586–96. <https://doi.org/10.18653/v1/N19-1361>.
- [2] Hyunji Park, Yogarshi Vyas, and Kashif Shah. 2022. “Efficient Classification of Long Documents Using Transformers.” In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 702–709, Dublin, Ireland. Association for Computational Linguistics.
- [3] Sami Uddin, Bipasha Banerjee, Jian Wu, William A. Ingram, Edward A. Fox. “Building A Large Collection of Multi-domain Electronic Theses and Dissertations”. 2021 IEEE International Conference on Big Data.
- [4] Pethe, Charuta, Allen Kim, and Steven Skiena. “Chapter Captor: Text Segmentation in Novels.” arXiv preprint arXiv:2011.04163 (2020).
- [5] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J. (2013). “Distributed Representations of Words and Phrases and their Compositionality.” Advances in neural information processing systems, 26.
- [6] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation.” In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532-1543. 2014.
- [7] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network.” arXiv preprint arXiv:1503.02531 2, no. 7 (2015).
- [8] Pappagari, Raghavendra, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. “Hierarchical Transformers for Long Document Classification.” In 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), pp. 838-844. IEEE, 2019.
- [9] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” arXiv preprint arXiv:1810.04805 (2018).
- [10] Beltagy, Iz, Matthew E. Peters, and Arman Cohan. “LongFormer: The Long-Document Transformer.” arXiv preprint arXiv:2004.05150 (2020).

## BIBLIOGRAPHY

- [11] Zaheer, Manzil, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham et al. “Big Bird: Transformers for Longer Sequences.” *Advances in Neural Information Processing Systems* 33 (2020): 17283-17297.
- [12] Le, Quoc, and Tomas Mikolov. “Distributed Representations of Sentences and Documents.” In *International conference on machine learning*, pp. 1188-1196. PMLR, 2014.
- [13] Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.” *arXiv preprint arXiv:1910.01108* (2019).
- [14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. “Attention Is All You Need”. *Advances in neural information processing systems*, 30.
- [15] Torres-Moreno, Juan-Manuel, ed. “Automatic Text Summarization”. John Wiley & Sons, 2014.
- [16] Jason Brownlee. “A Gentle Introduction to Text Summarization”. 2017. <https://machinelearningmastery.com/gentle-introduction-text-summarization>. Accessed on December 3, 2022.
- [17] Nallapati, Ramesh, Bowen Zhou, Caglar Gulcehre, and Bing Xiang. “Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond.” *arXiv preprint arXiv:1602.06023* (2016).
- [18] See, Abigail, Peter J. Liu, and Christopher D. Manning. “Get To The Point: Summarization with Pointer-Generator Networks” *arXiv preprint arXiv:1704.04368* (2017).
- [19] Ingram, William A., Bipasha Banerjee, and Edward A. Fox. “Summarizing ETDs with deep learning.” *Cadernos BAD* 1 (2019): 46-52.
- [20] Mihalcea, Rada, and Paul Tarau. “TextRank: Bringing Order into Texts.” In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pp. 404-411. 2004.
- [21] Erkan, Günes, and Dragomir R. Radev. “LexRank: Graph-based Lexical Centrality as Salience in Text Summarization.” *Journal of artificial intelligence research* 22 (2004): 457-479.
- [22] Luhn, Hans Peter. “The Automatic Creation of Literature Abstracts.” *IBM Journal of research and development* 2, no. 2 (1958): 159-165.
- [23] Dumais, Susan T. “Latent semantic analysis.” *Annu. Rev. Inf. Sci. Technol.* 38, no. 1 (2004): 188-230.

## BIBLIOGRAPHY

- [24] Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.” *J. Mach. Learn. Res.* 21, no. 140 (2020): 1-67.
- [25] Lewis, Mike, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension.” *arXiv preprint arXiv:1910.13461* (2019).
- [26] Palakh Mignonne Jude. “Increasing Accessibility of Electronic Theses and Dissertations (ETDs) Through Chapter-level Classification”. June 2020, MS thesis, Computer Science, Virginia Tech, Blacksburg, VA 24061, <http://hdl.handle.net/10919/99294>.
- [27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” *Advances in neural information processing systems* 32 (2019).
- [28] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, Alexander M. Rush. “HuggingFace’s Transformers: State-of-the-art Natural Language Processing.” *arXiv preprint arXiv:1910.03771* (2019).
- [29] Javaid Akbar Manzoor. “Segmenting Electronic Theses and Dissertations By Chapters.” Virginia Tech, Computer Science, MS thesis defended September 23, 2022, Blacksburg, VA 24061 USA, available late 2022 from <https://vtechworks.lib.vt.edu>.
- [30] Cornell University. “arXiv.org e-Print archive”. 2022. Retrieved December 3, 2022, from <https://arxiv.org>.
- [31] Asma Aldrees and Azeddine Chikh and Jawad Berri. “Comparative Evaluation Of Four Multi-label Classification Algorithms In Classifying Learning Objects”, 2016, Computer Science & Information Technology, 10.5121/csit.2016.60210.
- [32] Iz Beltagy and Arman Cohan and Kyle Lo. “SciBERT: Pretrained Contextualized Embeddings for Scientific Text”. 2019, CoRR, <http://arxiv.org/abs/1903.10676>.
- [33] Chin-Yew Lin. 2004. “ROUGE: A Package for Automatic Evaluation of Summaries.” In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

## BIBLIOGRAPHY

- [34] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. “BLEU: a method for automatic evaluation of machine translation.” In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL '02). Association for Computational Linguistics, USA, 311–318. <https://doi.org/10.3115/1073083.1073135>.
- [35] Zhang, Tianyi, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. “BERTScore: Evaluating Text Generation with BERT.” arXiv preprint arXiv:1904.09675 (2019).
- [36] Virginia Tech. “Computer Science Cloud at VT”. Retrieved December 3, 2022, from <https://cloud.cs.vt.edu/>.
- [37] Merkel, D. (2014). “Docker: lightweight Linux containers for consistent development and deployment”. Linux Journal, 2014(239).
- [38] Virginia Tech. “Virginia Tech GitLab”. 2022. Retrieved December 3, 2022, from <https://code.vt.edu/>.
- [39] Mišo Belica. “Automatic Text Summarizer”. 2022. Retrieved December 3, 2022, from <https://github.com/miso-belica/sumy>.
- [40] Hailei. “PDF Plumber Library Documentation”. 2020. Retrieved December 3, 2022, from [https://gitee.com/hailei\\_yan/pdfplumber](https://gitee.com/hailei_yan/pdfplumber).
- [41] Van Rossum, G., Drake, F. L. (2009). “Python 3 Reference Manual”. Scotts Valley, CA: CreateSpace.
- [42] Vingelmann, P., Fitzek, F. H. P. (2020). “CUDA release: 10.2.89”. Retrieved December 3, 2022, from <https://developer.nvidia.com/cuda-toolkit>.
- [43] Kenneth Reitz, Cory Benfield, Ian Stapleton Cordasco, Nate Prewitt et al. “Requests Library Documentation. Requests”. 2022. Retrieved December 3, 2022, from <https://requests.readthedocs.io/en/latest/>.
- [44] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. “TensorFlow: Large-scale machine learning on heterogeneous systems”, 2015. Software available from tensorflow.org.



## BIBLIOGRAPHY

- [45] Pedregosa, F., Varoquaux, Gael, Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). “Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*”, 12(Oct), 2825–2830.
- [46] McKinney, Wes. “Data Structures for Statistical Computing in Python.” *Proceedings of the 9th Python in Science Conference*. Vol. 445. No. 1. 2010.
- [47] J. D. Hunter, “Matplotlib: A 2D Graphics Environment” in *Computing in Science Engineering*, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55.
- [48] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., & Oliphant, T. E. (2020). “Array programming with NumPy”. *Nature*, 585(7825), 357-362.
- [49] Honnibal, Matthew, and Ines Montani. “spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.” January 2017, <https://sentometrics-research.com/publication/72/>
- [50] Bird, S., Klein, E., & Loper, E. (2009). “Natural language processing with Python: analyzing text with the natural language toolkit”. Reilly Media, Inc.
- [51] Mathieu Fenniak. “pyPdf 1.13”. 2010. Retrieved December 3, 2022, from <https://pypi.org/project/pyPdf/>.
- [52] Allen Institute for Artificial Intelligence. “Semantic Scholar”. 2022. Retrieved December 12, 2022, from <https://www.semanticscholar.org/>.
- [53] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., Kang, J. (2019). “BioBERT: a pre-trained biomedical language representation model for biomedical text mining”. *arXiv*. <https://doi.org/10.1093/bioinformatics/btz682>.
- [54] Kryściński, W., Rajani, N., Agarwal, D., Xiong, C., Radev, D. (2021). “BookSum: A Collection of Datasets for Long-form Narrative Summarization”. *arXiv*. <https://doi.org/10.48550/arXiv.2105.08209>.
- [55] Jordan Walke. “React A JavaScript library for building user interfaces”. 2022. Retrieved December 13, 2022, from <https://reactjs.org/>.
- [56] Lester, Brian, Al-Rfou, Rami, and Constant, Noah. (2021). “The Power of Scale for Parameter-Efficient Prompt Tuning”. “<https://aclanthology.org/2021.emnlp-main.243>”. “10.18653/v1/2021.emnlp-main.243”.