Classifying ETDs

CS4624 (Multimedia, Hypertext, and Information Access)
Instructor: Edward A. Fox
Virginia Tech, Blacksburg, VA 24061
5/17/2023

By: Vedant Shah, Mihir Gathani, Reema Daniel and Vaishali Ramesh
Client: Bipasha Banerjee
# Table of Contents

List of Figures ....................................................................................................................... 4
List of Tables .......................................................................................................................... 6

1.0 Abstract / Executive Summary ......................................................................................... 7

2.0 Introduction .......................................................................................................................... 8
  2.1 Background ......................................................................................................................... 8
  2.2 Objective ............................................................................................................................. 9
  2.3 Client .................................................................................................................................. 9
  2.4 Team Members and Roles ................................................................................................. 10

3.0 Deliverables/Requirements ............................................................................................... 11
  3.1 Deliverables ....................................................................................................................... 11
  3.2 Requirements .................................................................................................................... 11
    3.2.1 Data Cleaning/ Data Processing .................................................................................... 11
    3.2.2 Model Development .................................................................................................... 12
    3.2.3 Student Upload/ Website ............................................................................................ 12

4.0 Design ................................................................................................................................... 13
  4.1 Data ................................................................................................................................... 13
  4.2 Website Design .................................................................................................................. 13
  4.3 Deep Learning Model ........................................................................................................ 15

5.0 Implementation .................................................................................................................. 17
  5.1 Creation of Stand-Alone Website ...................................................................................... 17
  5.2 Data Analysis and Preprocessing metadata ..................................................................... 23
  5.3 CSV Input/output .............................................................................................................. 36
  5.4 Text Classification Model Fine-Tuning & Development .................................................. 37
  5.5 Coding Environment ........................................................................................................ 41

6.0 Testing/Evaluation/Assessment ......................................................................................... 42
  6.1 Website ............................................................................................................................. 42
  6.2 Data Extraction Process .................................................................................................... 42
  6.3 Text Classification Model Evaluation .............................................................................. 43

7.0 User’s Manual ................................................................................................................... 47
  7.1 Steps on how to use the website ...................................................................................... 47

8.0 Developer’s Manual .......................................................................................................... 52
  8.1 Jupyter Installation & Technology version check .............................................................. 52
  8.2 Data Cleaning and preprocessing .................................................................................... 52
  8.3 Website ............................................................................................................................. 54
  8.4 Text Classification Model ................................................................................................ 56
    8.4.1 Baselines ..................................................................................................................... 56
    8.4.2 Run Predictions .......................................................................................................... 58

9.0 Lessons Learned ................................................................................................................. 60
  9.1 Timeline/ Schedule ........................................................................................................... 60
List of Figures

Figure 1: Graphical representation of the distribution of ETDs across several departments............... 8
Figure 2: Wireframe of the Upload Page of the Website................................................................. 13
Figure 3: Wireframe of the Experimenter Page of the Website.................................................... 14
Figure 4: Wireframe of the Document View Page of the website.................................................. 15
Figure 5: Front-end Home page, which is the upload page........................................................... 17
Figure 6: Directory of our workbench. The red arrow shows the folder where the uploaded PDF and
text extraction is getting saved........................................................................................................ 18
Figure 7: Code snippet from the server.js file that triggers the text extraction code....................... 19
Figure 8: The public.inputeted table in the Postgres database...................................................... 19
Figure 9: The experimenter page of the website, where the selection of models is done and the
results are shown........................................................................................................................... 20
Figure 10: The snippet of the code written in server.js to trigger the machine learning code, which is
the run.py file.................................................................................................................................. 20
Figure 11: The public.outputted table in the database, where the results of the machine learning
models are stored........................................................................................................................... 21
Figure 12: The etdschema.chapter table column in the database, where the chapter details of an ETD
is stored........................................................................................................................................... 21
Figure 13: etd_id column of the chapter table.................................................................................... 22
Figure 14: Update query to add the E to the etd_id......................................................................... 22
Figure 15: The etdschema.classification table in the database, where the chapter labels and
additional details about the classification of ETDs are stored....................................................... 22
Figure 16: The etdschema.etd table in the database, where the details about the ETD are stored, such as author, department, and discipline............................................................... 22
Figure 17: The etdschema.mappingetd table in the database, where the label from machine learning
models and Proquest label are shown............................................................................................. 23
Figure 18: Search Query to display the information in a Dynamic Table in Document View Page...... 23
Figure 19: Output of all the columns in the metadata..................................................................... 24
Figure 20: Dropping some of the columns in preprocessing step.................................................... 25
Figure 21: Code for standardization................................................................................................ 25
Figure 22: removeBrackets() function code................................................................................... 25
Figure 23: Code for the createDept() method................................................................................. 26
Figure 24: Code for the renameDept() method............................................................................... 26
Figure 25: Code for merging similar departments......................................................................... 27
Figure 26: Mapping of STEM and Non-STEM.............................................................................. 28
Figure 27: Code creating two separate csv file for STEM and non STEM......................................... 29
Figure 28: Code to read CSV input.................................................................................................. 36
Figure 29: Code to save to a .csv file.............................................................................................. 36
Figure 30: A view of the Jupyter notebook with the input and output file..................................... 36
Figure 31: Tokenize Data for BERT-Based Models......................................................................... 38
Figure 32: Tokenize Data for Non-BERT based Models................................................................. 39
Figure 33: Modifying Non-BERT based models for text-classification..................................................40
Figure 34: Modifying BERT based Models for Text-Classification.........................................................40
Figure 35: Code for mapping ETD depts. with ProQuest Labels using Cosine Similarity..........................42
Figure 36: Predict Labels for Test Set.....................................................................................................44
Figure 37: Predict Multiple Probabilistic Labels for Input Data...............................................................45
Figure 38: Classification Report visualized as a Heat Map.......................................................................46
Figure 39: Upload your ETD page...........................................................................................................47
Figure 40: Choose the file needed...........................................................................................................48
Figure 41: Once the file is uploaded, it will display a “File Uploaded!”....................................................48
Figure 42: The Experimenter page which is the next landing page........................................................49
Figure 43: User can select the classification model..................................................................................49
Figure 44: Document View page............................................................................................................50
Figure 45: Dynamic table in Document View page................................................................................51
Figure 46: These are all the imports in the extract_text.py file, which are used to do text extraction. 55
Figure 47: These are all the imports in the run.py file, which are used to do machine learning models. 56
Figure 48: Running Inference Script (run.py) to Generate Predictions..................................................58
Figure 49: Checking Validity of Input Arguments....................................................................................59
Figure 50: react-scripts command not found error when trying to run npm start..................................61
Figure 51: Initialization error with React when trying to do npm start..................................................62
Figure 52: Error message refusing connection to our local host..............................................................62
Figure 53: pg-native error when trying to connect to the database.........................................................63
Figure 54: Proxy error when trying to do “npm start” to connect to the website....................................64
Figure 55: Invalid option object when trying to do “npm start”............................................................65
Figure 56: AxiosError once the user hits upload...................................................................................65
Figure 57: Upload or Database Connection error after the ETD file is uploaded.....................................66
Figure 58: Added lines of code in the package.json file to fix some of the initial error with using react-scripts........................................................................................................................................66
Figure 59: Dependencies in the package.json with the version number next to them..........................67
Figure 60: Reproducing Error for a Specific Text File on two Different Models......................................68
Figure 61: Tasks and Subtasks...............................................................................................................71
Figure 62: Workflow................................................................................................................................74
List of Tables

Table 1: Original Metadata (ETD_metadata.csv)........................................................................................................24
Table 2: Final_ETDMetadata after the data cleaning and preprocessing.................................................................30
Table 3: Final unique department list with Stem_NonStem label and count.........................................................31
Table 4: Top 25 STEM departments........................................................................................................................32
Table 5: Top 25 Non-STEM departments.................................................................................................................33
Table 6: ProQuest Subject Categories Dataset...................................................................................................34
Table 7: Mappings based on Cosine Similarity scores..........................................................................................35
Table 8: Mappings based on Cosine Similarity scores..........................................................................................36
Table 9: Table of Parameter Selection Heuristics.................................................................................................41
Table 10: Installation and Version Info..................................................................................................................52
Table 11: Data cleaning files and file type.............................................................................................................53
Table 12: Website files and file type.......................................................................................................................54
Table 13: Baseline Scores for all Fine-Tuned Models..........................................................................................58
Table 14: Timetable.................................................................................................................................................60
Table 15: Implementation of Service....................................................................................................................74
1.0 Abstract / Executive Summary

Electronic Theses or Dissertations (ETDs) are academic documents that provide an in-depth insight into and account of the research work of a graduate student, and are designed to be stored in machine archives and retrieved globally. These documents contain an abundance of information that may be utilized by various machine learning tasks such as classification, summarization, and question answering. However, these documents often have incomplete, incorrect, or inconsistent metadata which makes it challenging to accurately categorize these documents without manual intervention, since there is no one uniform format to develop the metadata. Therefore, through the Classifying ETDs capstone project, we aim to create a gold standard classification dataset, leverage machine learning and deep learning algorithms to automatically classify ETDs with missing metadata, and develop a website to allow a user to classify an ETD with missing metadata and view already classified ETDs. The expected impact of this project is to advance information availability from such long documents and eventually aid in improving long-document information accessibility through regular search engines.

To achieve our objectives, we worked with an existing dataset that contains basic ETD metadata. We created a gold-standard dataset for evaluation by cleaning the data, updating the ETD departments, and labeling the ETDs according to the ProQuest Subject Categories. In order to achieve automatic classification, we evaluated multiple categories of deep learning models like Encoder-Only based Transformer Models, Autoregressive Models, and Sequence-to-Sequence Models. Finally, we also created a stand-alone website to allow users to view already classified ETDs. The future work for the website would be to allow users to experiment with other unclassified ETDs.

Keywords: Gold Standard ETD Classification Dataset, Deep Learning, Text Classification Models, Interactive User Interface.
2.0 Introduction

2.1 Background

Our goal for this project is to be able to automate the process of classifying ETDs based on their corresponding departments, STEM or Non-STEM association, and ProQuest Subject Categories. Classification is the process of identifying and grouping objects or ideas into predetermined categories. An Electronic Thesis or Dissertation (ETD) is a document that expicates research and expresses it in a form simultaneously suitable for machine archives and worldwide retrieval. Electronic Theses or Dissertations (ETDs) contain important information that can be utilized in a wide variety of machine learning tasks such as classification. The motivation for our project is the creation of a gold standard dataset, and training deep learning models to classify all types of ETDs across the globe. The goal of the project is to make information from ETD’s more accessible to search engines.

Figure 1: Graphical representation of the distribution of ETDs across several departments
Some of the original problems that we were trying to solve were reducing from the number of unique departments to a smaller list of disciplines. From Figure 1, we can see that some names overlap, like for example, “Electrical & Computer Engr” and “Electrical and Computer Engineering”. These need to be grouped together and classified as one unique department with one unique name. Another issue we saw in the data was that some names were rather generic while other names were rather specific. For example, there is “Psychology” as a generic name, and “Educational Psychology” that’s very specific. Initially, we had 3150 unique disciplines and 1075 departments due to spelling mistakes, different nomenclature across universities, etc. Our goal was to reduce the unique departments as much as possible to create our gold-standard dataset. However, some of our new problems that we are trying to solve include standardizing the labels for ETD departments. Our other deliverables in this project include a high-precision text classification model and an interactive website for visualization. For the text classification model, we plan to fine-tune existing text classification models on our training dataset to empirically prove that automation of the classification process is indeed possible. We also intend to include an interactive UI so that the end-users can directly upload a document to a website and get the top three classification labels for the content uploaded. These custom labels can be mapped to standardized ProQuest labels. ProQuest labels have been developed after thoroughly researching multiple subjects over time and provide a good uniform labels base that can be cross-referenced anywhere in the world. (ProQuest ExLibris, 1938)

2.2 Objective

The objective of our project is to construct a gold standard dataset and a high-precision text classification model using the gold standard dataset. Additionally, the project goal is to create a standalone user-friendly website that allows users to upload their electronic thesis or dissertation, so it can classify data using the deep learning model developed.

2.3 Client

Our point of contact and client for this project is Ms. Bipasha Banerjee, a Ph.D. candidate and member of the Digital Library Research Laboratory at Virginia Tech (Virginia Tech), advised by Dr. Edward Fox. Her core research interests are Natural Language Processing, Data Mining, and Digital Libraries. Currently, she’s working on extracting and storing relevant information for long book-length documents such as Electronic Theses and Dissertations (ETDs). Our client is well-versed in the use of NLP techniques for the categorization of ETDs. Previously, she has published papers on ETD summarization (Ingram), classification and extraction of raw data from ETD documents (Aromando), and applications of data analytics on scholarly long documents (Banerjee). She’s been guiding us throughout the various phases of this project by providing us with reading material for topical context, raw data to get the project started, and a previously built website for us to build upon. Additionally, she’s also been helping us with deciding on natural language processing (NLP) models for baseline work. We have had a weekly meeting with her to update her on our progress and discuss the next steps for the subsequent week, to make good progress towards our milestones.
2.4 Team Members and Roles

- Mihir Gathani
  - Team and Data Lead.
  - R&D (Data Cleaning, Data preprocessing work, etc.)
  - Noting minutes of meetings.
  - Website Development.
- Vedant Shah
  - Research and Development (Building/ Implementing models, Analysis, etc.)
  - Point of Contact with Client.
  - Machine Learning Lead.
- Reema Daniel
  - Point of Contact with Dr. Fox.
  - Manual Annotation and Data Analysis.
  - Website Development.
  - Front-End Co-Lead.
- Vaishali Ramesh
  - Noting minutes of the meeting.
  - Data Cleaning and Website Development.
  - Front-End Lead.
3.0 Deliverables/Requirements

3.1 Deliverables

The deliverables for our project are:

1. A gold standard dataset extraction from the ETD_Metadata.csv will serve as a reliable source of training and testing data for the high-precision text classification model.
2. A high-precision text classification model capable of accurately classifying electronic thesis and dissertation data based on training with the gold standard dataset.
3. An interactive website that allows users to upload documents and classifies documents by utilizing the text classification model. This website will provide users with a usable interface for uploading their thesis and classifying their data.

Additionally, the deliverables also include any Jupyter or Google Collab Notebooks that are used to create the gold standard dataset.

3.2 Requirements

3.2.1 Data Cleaning/ Data Processing

Initially, we focused on building a data set that will be used as training data for the Machine Learning model. We were given metadata of ETDs and were told to focus on the Discipline and Department columns. For the process of data cleaning, we first standardized the data by mapping disciplines to the department column if the department was not provided, stripping any extra spaces in the data, applying spell check on the departments column, and changing all the department names to lowercase. Additionally, each ETD was assigned a unique ID consisting of a 5-6 digit number, with the first two digits representing the folder number for the corresponding ETD. However, we excluded ETDs with IDs starting from 52 or higher since we didn’t have to access those folders. We also removed the rows which had null values for title and abstract, and created a BasicCleanedETD.csv with the number of unique departments reduced from 1186 (original metadata that was given to us) to 1138. Next, we realized that there were a lot of inconsistencies in the names of the departments. For instance, some department names had unneeded brackets as part of the name, and sometimes the departments had unnecessary prefixes and suffixes in them, for instance, “department of computer science” or “computer science, college of”. We decided to ignore phrases like “department of”, “college of” or “school of” as we were concerned with the actual field (ex: “computer science”) and not the institution’s administrative divisions. Therefore, we mapped the departments to their parent departments. This process helped us reduce the number of unique departments to 516. We stored the resulting list in AdvancedCleanedETD.csv. After this, we found the number of documents within each unique department, and mapped each department to STEM and Non-STEM based on the DHS STEM Designated Degree Program List. Originally, we focused on finding 100 documents across 50 different disciplines/departments, more specifically, 25 Non-STEM and 25 STEM departments. However, post mid-course correction, we decided to select 200 documents across 25 STEM and 21 Non-STEM unique departments. Additionally, we also decided to map the ETD departments to ProQuest Subject Categories in order to better classify the departments. In order to do this mapping, we used a combination of cosine similarity and manual intervention.
3.2.2 Model Development

Once all the data was standardized and preprocessed, building an end-to-end deep learning pipeline required a few steps. These steps are defined under two broad categories: steps before mid-semester correction and ones after.

Pre-Mid-Semester Course Correction:
1) Collecting a training dataset out of the preprocessed total data such that the dataset would be balanced between top 25 STEM and 25 Non-STEM subjects, and there would only be 100 ETDs per subject, so 5000 documents in total.
2) Preprocessing this balanced dataset in a way that it could be fed to the pre-trained model as input (ex.: converting categorical labels to numeric values).
3) Returning a deep learning (DL) model, with the best model weights saved so that the process can be replicated by other researchers, which can classify ETD documents based on the title and abstract of the provided document.

Modifications Post Mid-Semester Course Correction:
1. Developing two separate .py files for training and inference, instead of a single .ipynb file, for running the end-to-end pipeline on a server instead of Google Colab (Google, 2017) environment to sidestep its GPU limitations.
2. Modifying the collected training dataset to ensure that we have the electronic theses and dissertations for all rows in the dataset, and it doesn’t include the segmented IDs provided to us by the client, so that we can use those in a testing dataset to evaluate each model’s performance on completely unseen data.
3. To better learn the underlying relations in the data, increase the size of the training data. Take 25 unique STEM and 21 unique Non-STEM subjects with 200 documents per subject resulting in a dataset with 9200 rows which is both relatively class-balanced and diverse, while accounting for data limitations on the clients’ end.
4. Preprocessing this dataset in a way that it could be fed to the pre-trained model as input (ex.: converting categorical labels to numeric values).
5. Returning a deep learning (DL) model, with best model weights saved so that the process can be replicated by other researchers, which can classify ETD documents based on the text of the uploaded document.

3.2.3 Student Upload/ Website

Initially, we wanted to use the client’s code base and modify the current working website to meet our functionality of uploading the ETDs and classifying them, and displaying the results. However, we encountered many problems using React, so we changed the requirements to do it on a local host. The PDF will be saved locally and have the text extracted and the inputs will be given to a machine learning model for classification. We are required to make 3 pages: the upload page, Experiment page, and Document View page. This is the GitLab repository link for the website code (Ramesh, 2023).
4.0 Design

4.1 Data
The ETD data that we were given contained significant inconsistencies. For instance, some ETDs lacked department names, and sometimes, the same department names were written in different forms (e.g., "Computer Science" and "computer sciences"). To address this issue, we first limited our dataset to only include the documents that we had access to. Next, we figured that in many of the instances where a department was missing, we had a discipline name available which could be used as a department name. Therefore, we decided to map discipline names to department names when the department name was not provided. Additionally, due to inconsistencies in naming, we decided to write a mapping function to map departments which were spelled differently but pointed to the same parent department.

4.2 Website Design
For the initial steps of the website design, we created wireframes of each of the pages and then started the coding for our front-end website. The pages we decided to create were Upload, Experimenter, and Document View.

![Figure 2: Wireframe of the Upload Page of the Website](image)
The Upload page of the website is the homepage where the PDF of the ETD is uploaded, as shown in Figure 2. The Experimenter page as shown in Figure 3 has 3 drop-downs where the user can select which machine learning model they want us to run to generate the labels. After the machine learning models are executed and the labels are produced, the results will be exhibited on a single page in the form of a table. The table will comprise the Title of the document, the name of the model utilized, and the corresponding labels generated by the model, along with their respective confidence scores.

Figure 3: Wireframe of the Experimenter Page of the Website

The Document View page of the website shows a dynamic table that displays all of the ETDs that are based on the keyword. The design is to search through the database and use the title and abstract to find ETDs that match, through keyword search. The wireframe for the Document view page showing all of the content that will be presented to the user is displayed in Figure 4.
4.3 Deep Learning Model

Four different deep learning models have been fine-tuned on the title and abstract of the electronic theses and dissertations from our old training dataset called modified_etd.csv with 5000 class-balanced rows representing 25 STEM and 25 Non-STEM subjects. Next, we’ll describe the design of the models in the context of why they were selected. Later in Section 8.4 we’ll mention the hyperparameters used to get the best results.

1. Encoder-Only based Transformer Models:
   a. We fine-tuned two Bidirectional Encoder Representations from Transformers (BERT) based domain-specific models viz. SciBERT and BioBERT (Beltagy et al., 2019) (Lee et al., 2019).
   b. The major benefit of using these models is that they’ve been pre-trained on a large corpus of scientific data (Beltagy et al., 2019) (Lee et al., 2019).
c. Thus, they can identify scientific terms and understand their context quite well. This can be very beneficial for STEM document classification and can help improve generalization performance on unseen new scientific documents.

2. Autoregressive Models:
   a. We fine-tuned the Generative Pre-trained Transformer (GPT) - 2 model belonging to this category of generative models (Radford et al., 2019).
   b. The major benefit of using this model is that it's been pre-trained on a large corpus of publicly available data crawled from the web.
   c. Additionally, unlike the BERT based models that have a maximum sequence length of 512; GPT-2 can have a larger sequence length so we used 1024 as the maximum sequence length (Radford et al., 2019). This allows the model to capture more contextual information when generating predictions.
   d. We believe that these factors, in addition to the diversity of data used for pre-training, can be useful when classifying different genres of electronic thesis and dissertation data.

3. Sequence-to-Sequence Models:
   a. We fine-tuned the BigBird-Pegasus model (a variation of the Pegasus model released by Google) belonging to this category of generative models (Zaheer et al., 2020).
   b. We used the BigBird-Pegasus model pre-trained on the Booksum dataset instead of the one released by Google because the Booksum dataset contains text data from a large variety of domains (Krzyścicki et al., 2022) which may help improve our generalization performance on unseen data.
   c. Another advantage of using this model is that its maximum sequence length is 4096, so it can capture a lot more context than the other models which can help improve the quality of predicted labels.
5.0 Implementation

5.1 Creation of Stand-Alone Website

Initially, our plan was to use the existing code base, which used node.js and React, to update the existing website (CS5604 Fall 2022 Class, 2022). But, we have encountered many problems such as database connection errors, file upload errors, and other issues, which are mentioned in Section 10.2. After extensively testing out possible solutions to get the website to work on our local host, and exhausting all our resources, the team decided to change approach, and try making a stand-alone website using node.js and bootstrap. The purpose of using Bootstrap is for beautifying the website.

![Figure 5: Front-end Home page, which is the upload page](image)

Figure 5 highlights the upload page, where the user can choose a PDF file and upload a chapter of an ETD. Once the user hits Upload, this will trigger the text extraction, and store the extracted text in the .txt file, refer to Figure 6. Both the submitted ETD(.pdf) and extracted text file(.txt) will be stored inside the uploads folder in the back end, and the information will be updated in the database, inside the input table.
Figure 6: Directory of our workbench. The red arrow shows the folder where the uploaded PDF and text extraction is getting saved.

The text extraction code is written in extract_text.py and gets called in our server.js file. If the ETD PDF file is successfully uploaded on the website, it will display a message saying “File Uploaded!”. This will trigger the text extraction code, which is shown in Figure 7.
Figure 7: Code snippet from the server.js file that triggers the text extraction code.

```javascript
res.send('File uploaded!');
// call python file text extraction
const { spawn } = require('child_process');

const filename = require('path').resolve(__dirname, 'uploads', 'filename.txt');
const pythonProcess = spawn('python', ['extract_text.py', 'uploads', filename]);

pythonProcess.stdin.write(JSON.stringify({
  functionName: 'process_files',
  arguments: ['uploads/', filename]
}));
```

Figure 8: The public.inputetd table in the Postgres database

<table>
<thead>
<tr>
<th>input_id</th>
<th>filename</th>
<th>extracted_text_path</th>
<th>filepath</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>file-168135751854.pdf</td>
<td>uploads/file-168135751854_cleanText.txt</td>
<td>uploads/</td>
</tr>
<tr>
<td>E2</td>
<td>file-1681357519013.pdf</td>
<td>uploads/file-1681357519013_cleanText.txt</td>
<td>uploads/</td>
</tr>
</tbody>
</table>

Once the code is extracted and saved as a .txt file, the values in the input table will get updated. The table stores the information about the input_id, which is set as an alphanumeric that starts with an “E” and then increments. The other columns include filename, text_file_path (the path of where the text file is saved) and the filepath (the path of the original PDF).

The next page of the website is the experimenter page, this is where the user chooses from the 3 classification models, as shown in Figure 6. Once the classification models are picked, then the machine learning code is triggered and the output of that is stored in the output table.
Once a model is selected, the next step is the machine learning code. That is written in the run.py routine, which is triggered. This trigger for the machine learning model is written in server.js, as shown in Figure 10. The code to trigger the machine learning has not been tested, since we needed a server to run the models in real-time and we didn’t have access to the same.

```javascript
//call machine learning code
const { exec } = require('child_process');

// Spawn a Python process
const Process = exec('python', ['run.py', 'bigbird', filename]);

// Send a message to the Python process
Process.stdin.write(JSON.stringify({
    functionName: 'main',
    arguments: ['bigbird', filename]
}));

// Handle the response from the Python process
Process.stdout.on('data', (data) => {
    const response = JSON.parse(data.toString());
    console.log(response.result);
});
```

Figure 10: The snippet of the code written in server.js to trigger the machine learning code, which is the run.py file.
All the output from running the machine learning models will be saved in the public.outputted table in the Postgres database, as shown in Figure 11. The table has a search_id column, which matches the input_id. The output_id column is filled in by using a procedure in Postgres where it’s the input_id concatenated with the ModelNo. For example, let’s say we have an input_id of 1121, and the user picks the first model as BigBird, then it would be 11211 saved in the output_id. For the next model, let’s assume the user picks SciBert, then it will be incremented, so 11212 for the output_id. The model_label is the classification label that was generated from running the machine learning model, and the model confidence is the corresponding confidence score for that label.

![database_table](image)

**Figure 11:** The public.outputted table in the database, where the results of the machine learning models are stored.

Next, we implemented the Document Views page as per the design. This page allows the user to search for ETDs based on the topic they input in the search bar. In the backend, we use the title and abstract, and display all the ETDs that correspond to the topic. To find the relevant ETDs we will use the chapter, classification, etd, and mappingetd tables from the Postgres database, which will already have the data on classified ETDs and their associated labels.

![database_tables](image)

**Figure 12:** The etdschema.chapter table column in the database, where the chapter details of an ETD is stored.
The etd_id column has an update query to add an E in front of the etd_id number. The query is shown in Figure 13.

```sql
UPDATE etdschema.chapter SET etd_id = 'E' || etd_id
WHERE etd_id NOT LIKE '%E%'
AND etd_id IS NOT NULL
;
```

Figure 14: Update query to add the E to the etd_id

Figure 15: The etdschema.classification table in the database, where the chapter labels and additional details about the classification of ETDs are stored

Figure 16: The etdschema.etd table in the database, where the details about the ETD are stored, such as author, department, and discipline.
Figure 17: The etdschema.mappingetd table in the database, where the label from machine learning models and Proquest label are shown.

All these tables will be used to display the data as a dynamic table in the Document View Page of the website. The search query that joins the table together and maps the information of each ETD and displays it in an dynamic table is shown in Figure 18.

Figure 18: Search Query to display the information in a Dynamic Table in Document View Page

The search query maps the ETD to the other tables using the etd_id column. The labels and scores in the classification table are mapped to the mappingetd table using the label column. The corresponding proquest_label is displayed for each ETD id. All the chapter information such as chapter_no and chapter_text is in the chapter table, which is why we also map those with the ETD. The query uses inner join, group by, concat and where conditions to get an efficient search query.

5.2 Data Analysis and Preprocessing metadata

The original data file was called ETD_metadata.csv, where there were initially 1186 unique departments. Table 1 shows the first portion of ETD_metadata.csv. Notice that the department and discipline columns are blank; this shows that there were a lot of null values, which we removed when making our cleaned dataset.
The first part of our implementation was performing some basic data analysis and then deciding what changes need to be made to clean the dataset. We used a Jupyter notebook (Banerjee, 2023) to perform the data analysis. We found information about each of the columns and its data type, using the .info() function in Pandas (McKinney et al.) to get a better understanding of what columns we should keep and what to drop. See Figure 19.

```
In [7]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 533847 entries, 0 to 533046
Data columns (total 15 columns):
# Column       Non-Null Count   Dtype
---          --------------   -----=
0 Unnamed: 0   533847 non-null int64
1 id           533847 non-null int64
2 title        533025 non-null object
3 author       532946 non-null object
4 advisor      413425 non-null object
5 year         467088 non-null object
6 abstract     432687 non-null object
7 university   522965 non-null object
8 degree       417669 non-null object
9 URI          532865 non-null object
10 department  287834 non-null object
11 discipline  365442 non-null object
12 language    488409 non-null object
13 schooltype  458033 non-null object
14 oadsclassifier  533847 non-null int64
dtypes: int64(3), object(12)
memory usage: 61.0+ MB
```

Figure 19: Output of all the columns in the metadata

After discussing with the client, for the preprocessing, Figure 20 shows the columns we decided to drop:

```
In [8]: df = df.drop(['oadsclassifier', 'language', 'author', 'advisor', 'Unnamed: 0'], axis=1)
```

Figure 20: Dropping some of the columns in preprocessing step
As a team, we decided on the following changes listed below (see also Figure 20) to clean the metadata set.

Changes:
- Dropped the unneeded columns
- Dropped rows with NaN values
- Make all the labels lower case
- Spaces (lstrip and rstrip)
- Spell Check on departments
- Excluded ETDs with IDs starting from 52 or higher since we didn’t have to access those folders. (Each ETD was assigned a unique ID consisting of a 5-6 digit number, with the first two digits representing the folder number for the corresponding ETD.)

```python
In [15]: def standardization(var):
    var = str(var).lower()
    var = var.lstrip()
    var = var.rstrip()
    return var

In [16]: df['title'] = df['title'].map(standardization)
df['abstract'] = df['abstract'].map(standardization)
df['university'] = df['university'].map(standardization)
df['department'] = df['department'].map(standardization)
df['discipline'] = df['discipline'].map(standardization)
```

**Figure 21:** Code for standardization

We also created a few functions to help reduce the unique department count such as `removeBrackets()` and `createDept()`, shown below in Figures 21 and 22, respectively.

```python
def removeBrackets(var):
    var = str(var)
    var = var.replace("[", """)
    var = var.replace("]", """)
    return var

df_1['new_dept'] = df_1['new_dept'].map(removeBrackets)
```

**Figure 22:** `removeBrackets()` function code

The code above removed brackets in the department, so that more departments could be classified as one.
The createDept() method removes prefixes like "College of" or "department of", so that for example, "College of computer science" and "Department of computer science" will have the same unique department name, "computer science". All the new department names will be saved in the Updated_Dept column.

Next, we performed some manual annotations and went through and found the departments that were incorrectly named and fixed those. See Figure 24.

Then, we implemented the code to merge similar departments. For example, a department named "electrical and computer engr" and "electrical and computer engineering" will be named "electrical and computer engr" in the Updated_Dept column, since they mean the same thing. And, the count (count of ETDs) for both the departments will be combined. The code is shown in Figure 25.
Then, we applied code to classify the department as STEM or Non-STEM on the cleaned data (see Figure 26). The code basically references the DHS STEM Designated Degree Program List STEM list, which was given by the client. (Department of Homeland Security [DHS], 2022) We labeled all the rows Non-STEM first then we assigned it STEM if it was on the STEM list using a simple method. For example, the code checks whether the department name has engineering in it, and if it does then labels it STEM.
Next, we created two separate CSV files, one STEM and one Non-STEM, which reads from the Final_UniqueDept_Count.csv and sorts in descending order, so we could use it for the 25 STEM and 25 Non-STEM counts. Refer to the code in Figure 27.
Figure 27: Code creating two separate csv file for STEM and non STEM

The image in Table 2 shows what the Final_ETDMetadata looks like after all the data cleaning and preprocessing is complete. Each ID has a newly updated_Dept column and Stem_NonStem column, which shows what department it is classified in and whether that comes under STEM or Non-STEM. In the end, we were able to get the data down to 513 unique departments.
Table 2: Final_ETDMetadata after the data cleaning and preprocessing

After all the data cleaning and preprocessing, Table 3: shows a summary of what we obtained for the unique departments, and each department's count.
Table 3: Final unique department list with Stem_NonStem label and count

<table>
<thead>
<tr>
<th>Updated_Dept</th>
<th>Stem</th>
<th>NonStem</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>electrical and computer engineering</td>
<td>Stem</td>
<td></td>
<td>3937</td>
</tr>
<tr>
<td>psychology</td>
<td>Stem</td>
<td></td>
<td>2276</td>
</tr>
<tr>
<td>mechanical engineering</td>
<td>Stem</td>
<td></td>
<td>2222</td>
</tr>
<tr>
<td>computer science</td>
<td>Stem</td>
<td></td>
<td>2086</td>
</tr>
<tr>
<td>music</td>
<td>NonStem</td>
<td></td>
<td>1738</td>
</tr>
<tr>
<td>chemistry</td>
<td>Stem</td>
<td></td>
<td>1727</td>
</tr>
<tr>
<td>english</td>
<td>NonStem</td>
<td></td>
<td>1394</td>
</tr>
<tr>
<td>physics</td>
<td>Stem</td>
<td></td>
<td>1229</td>
</tr>
<tr>
<td>mathematics</td>
<td>Stem</td>
<td></td>
<td>1193</td>
</tr>
<tr>
<td>curriculum and instruction</td>
<td>NonStem</td>
<td></td>
<td>1122</td>
</tr>
<tr>
<td>history</td>
<td>NonStem</td>
<td></td>
<td>1083</td>
</tr>
<tr>
<td>civil, architectural, and environmental engineering</td>
<td>Stem</td>
<td></td>
<td>1067</td>
</tr>
<tr>
<td>geological sciences</td>
<td>Stem</td>
<td></td>
<td>1026</td>
</tr>
<tr>
<td>educational psychology</td>
<td>Stem</td>
<td></td>
<td>1020</td>
</tr>
<tr>
<td>educational leadership</td>
<td>Stem</td>
<td></td>
<td>955</td>
</tr>
<tr>
<td>architecture</td>
<td>Stem</td>
<td></td>
<td>810</td>
</tr>
<tr>
<td>civil and environmental engineering</td>
<td>Stem</td>
<td></td>
<td>729</td>
</tr>
<tr>
<td>biology</td>
<td>Stem</td>
<td></td>
<td>722</td>
</tr>
<tr>
<td>chemical engineering</td>
<td>Stem</td>
<td></td>
<td>696</td>
</tr>
<tr>
<td>anthropology</td>
<td>NonStem</td>
<td></td>
<td>625</td>
</tr>
<tr>
<td>economics</td>
<td>Stem</td>
<td></td>
<td>591</td>
</tr>
<tr>
<td>education</td>
<td>Stem</td>
<td></td>
<td>560</td>
</tr>
<tr>
<td>petroleum and geosystems engineering</td>
<td>Stem</td>
<td></td>
<td>547</td>
</tr>
<tr>
<td>educational leadership, policy, and technology studies</td>
<td>Stem</td>
<td></td>
<td>543</td>
</tr>
<tr>
<td>materials science and engineering</td>
<td>Stem</td>
<td></td>
<td>541</td>
</tr>
<tr>
<td>sociology</td>
<td>NonStem</td>
<td></td>
<td>541</td>
</tr>
<tr>
<td>aerospace engineering</td>
<td>Stem</td>
<td></td>
<td>531</td>
</tr>
</tbody>
</table>
Table 4 is the list of the Top 25 STEM.

<table>
<thead>
<tr>
<th>Updated_Dept</th>
<th>Stem</th>
<th>Sum of Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>electrical and computer engineering</td>
<td>3937</td>
<td></td>
</tr>
<tr>
<td>psychology</td>
<td>2276</td>
<td></td>
</tr>
<tr>
<td>mechanical engineering</td>
<td>2222</td>
<td></td>
</tr>
<tr>
<td>computer science</td>
<td>2086</td>
<td></td>
</tr>
<tr>
<td>chemistry</td>
<td>1727</td>
<td></td>
</tr>
<tr>
<td>physics</td>
<td>1229</td>
<td></td>
</tr>
<tr>
<td>mathematics</td>
<td>1193</td>
<td></td>
</tr>
<tr>
<td>civil, architectural, and environmental engineering</td>
<td>1057</td>
<td></td>
</tr>
<tr>
<td>geological sciences</td>
<td>1026</td>
<td></td>
</tr>
<tr>
<td>educational psychology</td>
<td>1020</td>
<td></td>
</tr>
<tr>
<td>educational leadership</td>
<td>955</td>
<td></td>
</tr>
<tr>
<td>architecture</td>
<td>810</td>
<td></td>
</tr>
<tr>
<td>civil and environmental engineering</td>
<td>729</td>
<td></td>
</tr>
<tr>
<td>biology</td>
<td>722</td>
<td></td>
</tr>
<tr>
<td>chemical engineering</td>
<td>696</td>
<td></td>
</tr>
<tr>
<td>economics</td>
<td>591</td>
<td></td>
</tr>
<tr>
<td>education</td>
<td>560</td>
<td></td>
</tr>
<tr>
<td>petroleum and geosystems engineering</td>
<td>547</td>
<td></td>
</tr>
<tr>
<td>educational leadership, policy, and technology study</td>
<td>543</td>
<td></td>
</tr>
<tr>
<td>materials science and engineering</td>
<td>541</td>
<td></td>
</tr>
<tr>
<td>aerospace engineering</td>
<td>531</td>
<td></td>
</tr>
<tr>
<td>educational administration</td>
<td>444</td>
<td></td>
</tr>
<tr>
<td>biomedical engineering</td>
<td>407</td>
<td></td>
</tr>
<tr>
<td>cellular and molecular biology</td>
<td>349</td>
<td></td>
</tr>
<tr>
<td>earth and atmosphere sciences</td>
<td>346</td>
<td></td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td><strong>26554</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Top 25 STEM departments
Additionally, Table 5 gives the list of the Top 25 Non-STEM departments.

<table>
<thead>
<tr>
<th>Updated_Dept</th>
<th>Sum of Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>music</td>
<td>1738</td>
</tr>
<tr>
<td>english</td>
<td>1394</td>
</tr>
<tr>
<td>curriculum and instruction</td>
<td>1122</td>
</tr>
<tr>
<td>history</td>
<td>1083</td>
</tr>
<tr>
<td>anthropology</td>
<td>625</td>
</tr>
<tr>
<td>sociology</td>
<td>541</td>
</tr>
<tr>
<td>communication studies</td>
<td>394</td>
</tr>
<tr>
<td>radio-television-film</td>
<td>379</td>
</tr>
<tr>
<td>community and regional planning</td>
<td>365</td>
</tr>
<tr>
<td>business administration</td>
<td>365</td>
</tr>
<tr>
<td>government</td>
<td>360</td>
</tr>
<tr>
<td>educ policy, orgzn and leadership</td>
<td>319</td>
</tr>
<tr>
<td>advertising</td>
<td>304</td>
</tr>
<tr>
<td>journalism</td>
<td>304</td>
</tr>
<tr>
<td>geography</td>
<td>289</td>
</tr>
<tr>
<td>special education</td>
<td>282</td>
</tr>
<tr>
<td>theatre and dance</td>
<td>245</td>
</tr>
<tr>
<td>nuclear, plasma, and rad engr</td>
<td>221</td>
</tr>
<tr>
<td>social work</td>
<td>220</td>
</tr>
<tr>
<td>philosophy</td>
<td>209</td>
</tr>
<tr>
<td>art and design</td>
<td>183</td>
</tr>
<tr>
<td>art history</td>
<td>174</td>
</tr>
<tr>
<td>public affairs</td>
<td>173</td>
</tr>
<tr>
<td>spanish and portuguese</td>
<td>172</td>
</tr>
<tr>
<td>middle eastern studies</td>
<td>171</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td><strong>11632</strong></td>
</tr>
</tbody>
</table>

Table 5: Top 25 Non-STEM departments

After the mid-semester course correction, we realized that there already existed a somewhat universal way of classifying departments using the ProQuest subject categories. Therefore, we decided to map the current Updated_Depts to the ProQuest subject categories. To achieve the mapping we first needed to create a dataset containing the ProQuest labels. In order to do this, we used a combination of the import feature in Excel and the mapping functionality of Python. (See a sample of the ProQuest dataset in Table 6.)
Once we had the ProQuest Mapping done, we needed to map the ETD departments to the ProQuest departments. To do this, we first tried to directly use a merge operation. However, this caused 417 departments of the total of 516 unique departments to remain unmapped. Therefore, we decided to use cosine similarity to aid with mapping of the ETD and ProQuest department names/labels. (See sample successful mappings done by cosine similarity in Table 7.)
As seen in Table 7, even after performing cosine similarity to aid with the mapping process, we could not map all 516 departments. Additionally, as seen in ID 285 of Table 7, we only had a singular mapping for Computer Science and Engineering, when we should have had the mapping as “Computer Science, Computer Engineering”. Therefore, we decided that we would restrict ourselves to only the previously selected 46 departments. The 46 departments consisted of TOP 25 STEM departments, and TOP 23 Non-STEM departments. The reason for the unequal STEM and Non-STEM department was that we needed a minimum of 200 documents per department. Next, we defined a dictionary with the required mappings for ETD departments to ensure proper mappings.

After this, we defined a mapping function to map the ProQuest departments names and to ensure that we also mapped all the corresponding department codes, labels, and categories. As you can see in Table 8, we mapped the ETD department “civil architectural and environmental engineering” to all 3 of “civil engineering, architectural engineering and environmental engineering.” We also ensured that the code, labels, and categories were also mapped appropriately.
Table 8: Mappings based on Cosine Similarity scores

5.3 CSV Input/output

Our input file is called “ETD_metadata.csv”. Then, as shown in Figure 28, we use the .read_csv function in Pandas to save it to a variable, so we can use it.

```python
df = pd.read_csv("ETD_metadata.csv")
```

Figure 28: Code to read CSV input

To save the output as a .csv file we are using the to_csv function as shown in Figure 29.

```python
df_nonstem.to_csv('NonStem.csv')
```

Figure 29: Code to save to a .csv file
Figure 30 shows the Jupyter notebook, where we created a folder for this Capstone class. When we created an output file, it was added here. We created these same “Updated_Department_Count” CSV files, which is what the red arrows are pointing to. The .ipynb is where we write the data analysis and data processing code.

5.4 **Text Classification Model Fine-Tuning & Development**

Developing a deep learning model to classify electronic theses and dissertations (ETDs) based on title, abstract, and document text posed an intriguing challenge. In this section we’ll describe the development of models agnostic of whether they were trained on title and abstract information or full ETD text because the process is exactly the same.

We began by researching popular natural language processing (NLP) models recommended by the client, such as BioBERT (Lee et al., 2019) and SciBERT (Beltagy et al., 2019). SciBERT showed the best results, but as with all BERT-based models, the maximum sequence length is limited to 512, which in turn limits the captured context. One way to capture more context was to increase maximum character length thereby using a bigger pre-trained model for longer sequences. To achieve this, we plotted a kernel density plot to determine the maximum character length of the documents in our dataset, thus ensuring uniformly padded encoded text for the model to train on.

Based on the analysis of our data using the kernel density plots described above, we realized that even just title and abstract combination had more than 2000 words, and of-course this would imply an automatic increase in this count when the entire document text is considered, thus, we decided to use models like GPT-2 and BigBird-Pegasus which have a relatively higher maximum sequence length and can capture more context when making a prediction (Radford et al., 2019) (Zaheer et al., 2020). Since all these models expect tokenized data as input, we wrote custom functions to preprocess all the data from the dataset seen in Figures 31, 32 respectively.
class ETDDataset_BERT(Dataset):

def __init__(self, total_text, target, tokenizer, max_len): #target = Updated_Dept
    self.total_text = total_text
    self.target = target
    self.tokenizer = tokenizer
    self.max_len = max_len

def __len__(self):
    return len(self.total_text)

def __getitem__(self, item):
    total_text = str(self.total_text)[item]
    # target = self.target[item]

    encoding = self.tokenizer.encode_plus(
        total_text,
        max_length=self.max_len,
        truncation=True, #changed
        add_special_tokens=True,
        padding='max_length', #changed
        return_attention_mask=True,
        return_token_type_ids=False,
        return_tensors='pt'
    )

    return {
        'total_text': total_text, # helpful for evaluating the model
        'input_ids': encoding['input_ids'].flatten(),
        'attention_mask': encoding['attention_mask'].flatten(),
        'targets': torch.tensor(self.target[item], dtype=torch.long) # targets are numeric type
    }
The major reasons for selecting each of the models have been described in more detail in the Section: 4.3 while the best hyperparameters for each of the models have been shown in the Section: 8.4. The different hyperparameters used regarding the optimizer and scheduler were selected for each of the models based on some generic heuristics of best implementations obtained from Hugging Face (see table 9), the respective papers where these models were originally implemented, and running multiple experiments on Google Colab. Since none of these models were originally built for text classification; rather for other natural language processing (NLP) tasks such as text summarization, question answering, or masked language modeling (Lee et al., 2019)(Beltagy et al., 2019) (Radford et al., 2019) (Zaheer et al., 2020); the last layer of all models was modified to add a drop out layer and final classification layer as seen in Figures 33 and 34.
The development of an end-to-end pipeline needs to consider two more major factors. One is data preprocessing. The other involves modifying the model's by employing a custom training loop and a function to evaluate and visualize the fine-tuned models quality of predictions, based on metrics such as accuracy, f1-score, precision, and recall. Additionally, gradient clipping and early callback techniques were employed to use the allocated resources as efficiently as possible. Gradient clipping essentially helps avoid the exploding gradients problem which can destabilize the model during training. The training loop also uses checkpoints so that if the model disconnects at any point due to any reason such as GPU limitations the best version of the model is saved so it can be re-initialized and trained from that point. Figures of the entire code haven’t been included in this report for the same reason as they are quite big; however, the code has been provided on a GitLab repository (Banerjee, 2023).

Finally, for evaluating the model, a custom get_multiple_predictions function is used to generate three probabilistic labels for each document provided as input. A snapshot of that code is provided along with a classification report that is generated using Sklearn’s classification_report method to better understand how the models perform on specific classes in the training_data and where they may require more attention. This can help improve training in future iterations. This code for the pipeline can be found in Section 6.3.
5.5 Coding Environment

We used Jupyter-based environments for both data cleaning, preprocessing, and creation of a gold standard dataset as well as building the machine learning model in terms of Jupyter notebooks and Google Colab respectively (Google, 2017). Google Colab is essentially a Jupyter environment with a wrapper on top of it and a backend hosted on Google Cloud. In terms of the data preprocessing and creation of a gold standard dataset, we used Python via the Jupyter notebooks, and specifically made use of the Pandas (McKinney et al.), NumPy (Harris et al., 2020), Sklearn (Pedregosa et al., 2011), Seaborn (Waskom et al., 2017), and TextBlob (Loria et al., 2018) libraries. Next, we used Google Colab for developing results from pre-trained models mentioned in the implementation because of easy access and a fast, free cloud-based environment integrated with Google Drive. This also makes accessing the data and saving the model easier. Additionally, many libraries we needed to use could be easily installed just using the pip command in Colab as opposed to installing it in the proper directories or on the server in Anaconda. However, in the long term for the final models, it's likely we'll need access to Advanced Research Computing (ARC) resources; thus, we've built an .ipynb file for training and a .py file for inference in Visual Studio (Microsoft Corporation, 2019) for the same. In terms of libraries, we used the Transformers library from Hugging Face (Hugging Face Community Contributors, 2019) as well as PyTorch (Paszke et al., 2019), Pandas, Numpy, Seaborn, Matplotlib (Hunter, 2007), and Sklearn. A more detailed description of these libraries in terms of versions is available in the Section 8.1. Initially, our plan was to use React (Walke et al., 2012), Node.js (Dahl, 2009), and PostgreSQL (The PostgreSQL Global Development Group, 1996) but we ran into a lot of problems like versions, pg-native errors, etc. refer to Section 10.2 Problems and Section 10.3 Solutions section. Therefore for the website, we used a combination of various technologies, including Node.js(Dahl, 2009), Bootstrap (Otto and Thornton, 2011), Express (Holowaychuk, 2010), pdf-parse(pdf-parse, 2023), Mutor (Walter Rumsby, 2010), and PostgreSQL (The PostgreSQL Global Development Group, 1996). We used Node.js because it’s powerful and allows us to develop fast and scalable web applications. For beautification, we used the Bootstrap framework which provides us with a set of CSS and JavaScript components for building responsive websites. We are also using Express, a flexible web application framework for Node.js that provides a robust set of features for websites. Additionally, we used several packages like pdf-parse, a Node.js package that provides an API for extracting text and other pieces of information from PDF files, and Mutor for uploading a file and handling a file. And integrating the Postgres database with UI provides us with powerful data management capabilities. All of these technologies helped us to build web applications with advanced features and functionality.

<table>
<thead>
<tr>
<th>Model</th>
<th>Heuristic Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>BioBert</td>
<td>(Lee et al., 2019) (Sun et al., 2019) (Devlin et al., 2018)</td>
</tr>
<tr>
<td>SciBert</td>
<td>(Beltagy et al., 2019) (Devlin et al., 2018)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>(Radford et al., 2019) (Korbak, 2022)</td>
</tr>
<tr>
<td>BigBird-Pegasus</td>
<td>(Zaheer et al., 2021)</td>
</tr>
</tbody>
</table>

Table 9: Table of Parameter Selection Heuristics
6.0 Testing/Evaluation/Assessment

6.1 Website
The evaluation we plan on doing is to make sure an user can successfully upload the ETD to the website and that it should be classified correctly. Another assessment is making sure the PDF extracted information is added to the input table in the backend. This way we can test the integration of the frontend and the backend when evaluating the website. The testing of the website is not complete, since we don’t have the website up on the server; refer to Section 10.4.

6.2 Data Extraction Process
In order to evaluate the data cleaning and preprocessing process, we first ran a spell checker on the departments associated with each ETD. However, the spell checker caused certain departments to be renamed into departments that did not make sense. For instance, “educ policy, orgzn and leadership” was renamed to “duc policy, organ and leadership”. Therefore, we had to manually find such departments and rename them based on their original departments. To ensure accuracy in this process, two of us went through the departments and decided which one should be renamed, and then one other team member checked them before the client herself confirmed it. Additionally, in order to aid with mapping the ETD data to the ProQuest Subject categories, we performed mapping based on cosine similarity scores (see the code in Figure 29). However, this too required manual intervention to ensure accuracy. To do this, the client and two team members manually confirmed whether the mappings were correct.

```python
import numpy as np

def cal_similarity(A, B):
    vectorizer = CountVectorizer()
    A = vectorizer.fit_transform(A)
    B = vectorizer.transform(B)
    res = cosine_similarity(A, B)
    return res

similarity_matrix = cal_similarity(df_ProQuest.Dept, df.UniqueETDepts.Updated_Dept)

max_similarities = similarity_matrix.max(axis=0)
threshold = 0.7 # tune this threshold to control what all is considered as a match
matches = [(df.ProQuest.index[idx1], df.UniqueETDepts.index[idx2], similarity) for idx2, (idx1, similarity) in enumerate(zip(similarity_matrix.argmax(axis=0), max_similarities)) if similarity >= threshold]

no_matches = [(df.ProQuest.index[idx1], df.UniqueETDepts.index[idx2], similarity) for idx2, (idx1, similarity) in enumerate(zip(similarity_matrix.argmax(axis=0), max_similarities)) if similarity < threshold]

matches_df = pd.DataFrame(columns=['ETD_Depts', 'ProQuest_Depts', 'similarity'])
no_matches_df = pd.DataFrame(columns=['ETD_Depts', 'ProQuest_Depts', 'similarity'])

# print the matches
matches_df = matches_df.append([{'ETD_Depts': df.ProQuest.loc[match[0], 'Dept'], 'ProQuest_Depts': df.ProQuest.loc[match[1], 'Dept'], 'similarity': match[2]], ignore_index=True])

for match in no_matches:
    no_matches_df = no_matches_df.append([{'ETD_Depts': df.ProQuest.loc[match[0], 'Dept'], 'ProQuest_Depts': df.ProQuest.loc[match[1], 'Dept'], 'similarity': match[2]], ignore_index=True])
```

Figure 35: Code for mapping ETD depts. with ProQuest Labels using Cosine Similarity
6.3 Text Classification Model Evaluation

We developed an end-to-end approach to evaluate the performance of our fine-tuned text classification models. To achieve this, we built a "get predictions" function, shown in Figure 36. This function returns the original text of test data from the data frame, the top predicted label, the prediction confidence score (probability), and the real label of the data. We ensured that the data_loader used in this function contains multiple rows that are diverse and representative of the original dataset during the testing process.

Using the output obtained from the "get predictions" function, we generated a detailed classification report. In addition, we developed code to create a confusion matrix that is visualized as a heatmap (refer to Figure 38 for an example). This allows us to easily visualize the performance of the model for each specific class and identify any patterns or trends. Additionally, the classification report returns precision, recall, and weighted f1-score which helps us learn more about how the model performs on each class of the data allowing us to improvise data or apply penalized data from certain classes as necessary.

To make our system more user-friendly, we also implemented a "get multiple predictions" function. This feature allows the user to obtain the top three probabilistic labels predicted by the model for a given input of text data (refer to Figure 37 for function code and Figure 48 in Section 8.4 for an example). Overall, our approach provides a comprehensive and detailed evaluation of our text classification models, which can help us identify areas for improvement and optimize the performance of our system.
```python
def get_predictions(model, test_data_loader, device):
    model = model.eval()

    review_texts = []
predictions = []
prediction_probs = []
real_values = []

    with torch.no_grad():
        for ex in test_data_loader:
            texts = ex["total_text"]
            input_ids = ex["input_ids"].to(device)
            attention_mask = ex["attention_mask"].to(device)
            target = ex["targets"].to(device)

            outputs = model(
                input_ids = input_ids,
                attention_mask = attention_mask
            )

            _, preds = torch.max(outputs, dim=1) #class with highest probability for each example

            probs = F.softmax(outputs, dim=1)

            review_texts.extend(texts)
predictions.extend(preds)
prediction_probs.extend(probs)
real_values.extend(target)

    predictions = torch.stack(predictions).cpu()
prediction_probs = torch.stack(prediction_probs).cpu()
real_values = torch.stack(real_values).cpu()

    return review_texts, predictions, prediction_probs, real_values
```

Figure 36: Predict Labels for Test Set.
```python
def get_multiple_predictions(model, data_loader, label_dict, device):
    model.eval()
    predictions = []
    prediction_probs = []
    review_text = []

    with torch.no_grad():
        for ex in data_loader:
            texts = ex['total_text']
            input_ids = ex['input_ids'].to(device)
            attention_mask = ex['attention_mask'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            softmax_outputs = torch.softmax(outputs, dim=1)

            # get the top 3 predicted labels and their corresponding probabilities
            _, indices = torch.topk(softmax_outputs, k=3, dim=1)
            labels = [[label_dict[i].item()] for i in indices_row for indices_row in indices]
            probs = [[softmax_outputs[j][i].item()] for i in indices_row for j, indices_row in enumerate(indices)]

            predictions.extend(labels)
            prediction_probs.extend(probs)
            review_text.extend(texts)

    return review_text, predictions, prediction_probs
```

Figure 37: Predict Multiple Probabilistic Labels for Input Data.
Figure 38: Classification Report visualized as a Heat Map.
7.0 User’s Manual

7.1 Steps on how to use the website
Website link: http://localhost:3000/
Step 1: Click on the Choose File button to upload your ETD.

![Choose File](image)

Figure 39: Upload your ETD page.

Step 2: Choose the PDF file you want to upload to get the classification. Then, click the open button as shown in Figure 40.
Step 3: Once the file is uploaded, it will display a “File Uploaded!” message and the user will automatically be taken to the next page, which is the Experimenter page.
Step 4: On the Experimenter page, the user has a drop-down to pick the machine learning models they want to run.

Figure 43: User can select the classification model
Step 5: When the user clicks on the drop-down bar, these are the options they will see. They can select the option they want to run.

Step 6: Displays the results on the Experimenter page as shown on the wireframe in the 4.2 Design section of the report (see Figure 44). This step has not been implemented because due to time constraints we were not able to integrate the machine learning models in the server. However, in our backend we have the output table, which will have the machine learning model results. Due to a time crunch, we have decided it is for Future Works (read more about this in Section 10).

![Figure 44: Document View page](image)

Step 7: The user can navigate to the Document View page by selecting “Document View” on the header. The page has a search bar to type in what ETD’s the user wants to view.
Step 8: The results of the search keyword will be displayed as a dynamic table in the Document View page.

Note, all the website features are shown in the video “ClassifyingETDsWebsiteDemo.mp4”. This can be a useful resource for users to familiarize themselves with the website and its functionality, as well as to learn how to navigate the different features available on the website.
8.0 Developer’s Manual

8.1 Jupyter Installation & Technology version check

The installation process for the Jupyter Notebook can take place in one of two ways. These are using a pip command or downloading the Anaconda application (Wang et al., 2012) on Windows and installing Jupyter Notebook and Lab from there using a GUI interface. Detailed instructions for installation using the GUI interface of Anaconda are provided in (Abhinav, 2021). Since time was of the essence and most students already had Anaconda preinstalled, we decided to use the GUI interface to install Jupyter Lab on our laptops. Please note that this model was developed in the Google Colab environment and not on a locally hosted GPU. Further, if any of the functions used in the model training code are deprecated at a later time, please execute the following line of code, with appropriate changes, to install the versions we had used at the time of original development, as listed in Table 9.

```bash
pip install --force-reinstall 'tool-name' == 'version_number'
ex: pip install --force-reinstall numpy == 1.22.4
```

<table>
<thead>
<tr>
<th>Installation</th>
<th>Version Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyTorch</td>
<td>1.13.1</td>
</tr>
<tr>
<td>Transformers</td>
<td>4.26.1</td>
</tr>
<tr>
<td>Sklearn</td>
<td>1.0.2</td>
</tr>
<tr>
<td>Seaborn</td>
<td>0.11.2</td>
</tr>
<tr>
<td>Pandas</td>
<td>1.3.5</td>
</tr>
<tr>
<td>Numpy</td>
<td>1.22.4</td>
</tr>
<tr>
<td>TextBlob</td>
<td>0.9.0</td>
</tr>
</tbody>
</table>

Table 10: Installation and Version Info

8.2 Data Cleaning and preprocessing
<table>
<thead>
<tr>
<th>File Name</th>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETD_Metadata</td>
<td>.csv</td>
<td>The file contains the metadata of the thesis like title, department, etc.</td>
</tr>
<tr>
<td>Stem</td>
<td>.csv</td>
<td>This file contains the list of Non-STEM and Count</td>
</tr>
<tr>
<td>NonStem</td>
<td>.csv</td>
<td>This file contains the list of STEM and Count</td>
</tr>
<tr>
<td>03162023_Analysis_ETD_Update</td>
<td>.ipynb</td>
<td>This file contains the Python code to generate a dataset containing ETDs based on the top 46 departments with consistent department labels</td>
</tr>
<tr>
<td>0320_Cleaned_ETDMetadata</td>
<td>.csv</td>
<td>This file contains the cleaned ETD Metadata with STEM / Non-STEM labeling</td>
</tr>
<tr>
<td>0320_Final_UniqueDept_Count</td>
<td>.csv</td>
<td>This file contains the count for no. of documents in each department and the STEM / Non-STEM labeling</td>
</tr>
<tr>
<td>06April2023_ProQuestLabel_Mapping</td>
<td>.ipynb</td>
<td>This file contains the Python code to perform mapping of ETD Departments to ProQuest Labels</td>
</tr>
<tr>
<td>BasicCleanedETD</td>
<td>.csv</td>
<td>This file contains the ETD data after basic standardization</td>
</tr>
<tr>
<td>AdvancedCleanedETD</td>
<td>.csv</td>
<td>This file contains the ETD data with consistent department names</td>
</tr>
<tr>
<td>0406_FinalDataset</td>
<td>.csv</td>
<td>This file contains all of the ETDs Documents from the Top 46 Depts.</td>
</tr>
<tr>
<td>0406_TrainingData</td>
<td>.csv</td>
<td>This file contains 200 documents from each of the Top 46 Depts., ensuring that no already segmented documents exist in this list</td>
</tr>
<tr>
<td>ProQuestDataset</td>
<td>.csv</td>
<td>This file contains the ProQuest Labels and associated data</td>
</tr>
<tr>
<td>Top46_Mappings</td>
<td>.csv</td>
<td>This file contains the mappings of the top 46 Depts. along with their associated ProQuest Labels</td>
</tr>
</tbody>
</table>

Table 11: Data cleaning files and file type
Table 11 shows the files required to create the gold standard dataset. It also contains the required input files required for the processing and mapping of data.

8.3 Website

Code overview of our stand-alone website. See Table 12.

<table>
<thead>
<tr>
<th>File Name</th>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server.js</td>
<td>.js</td>
<td>This file has the database connection and the triggers for text extraction, python code, and search query.</td>
</tr>
<tr>
<td>package.json</td>
<td>.json</td>
<td>This file contains all the versions and dependencies of our website.</td>
</tr>
<tr>
<td>package-lock.json</td>
<td>.json</td>
<td>This file lists all the packages and dependencies.</td>
</tr>
<tr>
<td>index.html</td>
<td>.html</td>
<td>This file has the code for the upload feature which acts as a homepage for our website.</td>
</tr>
<tr>
<td>extract_text.py</td>
<td>.py</td>
<td>This file has the code to do text extraction on the PDF file that is uploaded.</td>
</tr>
<tr>
<td>experimenter.html</td>
<td>.html</td>
<td>This file has code displays the Experimenter page where the dropdowns with the machine learning model listed</td>
</tr>
<tr>
<td>documentview.html</td>
<td>.html</td>
<td>Displays the dynamic table in Document View Page.</td>
</tr>
<tr>
<td>run.py</td>
<td>.py</td>
<td>This file has the code to take the inputs from the input table, run the Python code to run the machine learning model, and gives the results that will be added to the output table</td>
</tr>
<tr>
<td>etdschema_final_backup.sql</td>
<td>.sql</td>
<td>The SQL dump of the etdschema which is used for the Document View page.</td>
</tr>
<tr>
<td>input_output_final_backup.sql</td>
<td>.sql</td>
<td>The SQL dump of the input and output table which is used for the Upload page.</td>
</tr>
</tbody>
</table>

Table 12: Website files and file type
Table 12 contains all the corresponding files for the website. Some of the key frontend files are experimenter.html, index.html, and documentview.html, which are the pages that the user navigates through on the website. The etdschema_final_backup.sql and input_output_final_backup.sql files contain SQL dumps with the schemas for all the tables that are used to store information about the ETDs and the classification. The server.js file is used to connect the frontend to the database.

```
# Python Code for pdf to text extraction
import glob
import random
import os
import pandas as pd
import json
import pdfplumber
from tqdm import tqdm
import multiprocessing
from functools import partial
from timeit import default_timer as timer
import sys
```

Figure 46: These are all the imports in the extract_text.py file, which are used to do text extraction.

Before starting the website, make sure to do “pip3 install glob” and continue to install all the modules in order, using the same command.
Figure 47: These are all the imports in the run.py file, which are used to do machine learning models.

To run the Python file, first make sure to do “pip3 install torch” and continue to install all the modules in order, using the same command.

8.4 Text Classification Model

8.4.1 Baselines

Table 4 shows the baseline results for text-classification models fine-tuned on two categories of data. Category1: ETD title + abstract information, Category2: Full non-segmented ETD text.
<table>
<thead>
<tr>
<th>Base Model Name</th>
<th>Hyperparameter</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>SciBERT</td>
<td>Epochs = 10&lt;br&gt;Max_seq = 512&lt;br&gt;Learning_rate = 1e-5&lt;br&gt;Dropout = 0.3&lt;br&gt;Optimizer = AdamW&lt;br&gt;Weight_decay = 0.01&lt;br&gt;Scheduler = get_linear_schedule_with_warmup()&lt;br&gt;Num_warmup_steps = 10&lt;br&gt;Loss = Cross-Entropy Loss&lt;br&gt;Batch_Size = 4</td>
<td>Category 1:&lt;br&gt;F1-Score: 72.4%&lt;br&gt;Category 2:&lt;br&gt;F1-Score: 78.4%&lt;br&gt;Model_path = scibert_model_state.pth</td>
</tr>
<tr>
<td>BioBERT</td>
<td>Epochs = 10&lt;br&gt;Max_seq = 512&lt;br&gt;Learning_rate = 2e-5&lt;br&gt;Dropout = 0.3&lt;br&gt;Optimizer = AdamW&lt;br&gt;Weight_decay = 0.01&lt;br&gt;Scheduler = get_linear_schedule_with_warmup()&lt;br&gt;Num_warmup_steps = 10&lt;br&gt;Loss = Cross-Entropy Loss&lt;br&gt;Batch_Size = 4</td>
<td>Category 1:&lt;br&gt;F1-Score: 72.4%&lt;br&gt;Category 2:&lt;br&gt;F1-Score: 74.4%&lt;br&gt;Model_path = biobert_model_state.pth</td>
</tr>
<tr>
<td>GPT-2</td>
<td>Epochs = 7&lt;br&gt;Max_seq = 1024&lt;br&gt;Learning_rate = 6e-5&lt;br&gt;Dropout = 0.3&lt;br&gt;Optimizer = AdamW&lt;br&gt;Weight_decay = 0.1&lt;br&gt;Scheduler = CosineAnnealingWarmRestarts()&lt;br&gt;Epsilon = 1e-08&lt;br&gt;Betas = (0.9, 0.98)&lt;br&gt;Num_warmup_steps = int(total_steps * 0.01)&lt;br&gt;Loss = Cross-Entropy Loss&lt;br&gt;Batch_Size = 4</td>
<td>Category 1:&lt;br&gt;F1-Score: 73.8%&lt;br&gt;Model_path = gpt2_model_state.bin</td>
</tr>
<tr>
<td>BigBird-Pegasus</td>
<td>Epochs = 10&lt;br&gt;Max_seq = 4096&lt;br&gt;Learning_rate = 5e-5&lt;br&gt;Dropout = 0.3&lt;br&gt;Optimizer = AdamW&lt;br&gt;Weight_decay = 0.001&lt;br&gt;Scheduler = get_linear_schedule_with_warmup()&lt;br&gt;Epsilon = 1e-08&lt;br&gt;Betas = (0.9, 0.98)&lt;br&gt;Num_warmup_steps = int(total_steps * 0.1)&lt;br&gt;Loss = Cross-Entropy Loss&lt;br&gt;Batch_Size = 4</td>
<td>Category 1:&lt;br&gt;F1-Score: 73.4%&lt;br&gt;Model_path = bigbird_pegasus.pth</td>
</tr>
</tbody>
</table>
Table 13: Baseline Scores for all Fine-Tuned Models

NOTE: All of these models were trained with limited computational capacity. The BigBird model wasn’t even trained to completion as it ran out of GPU resources on ARC as well. Further only two of the four models were trained on both categories of data.

8.4.2 Run Predictions

Figure 48: Running Inference Script (run.py) to Generate Predictions.

In Figure 48, we see the process to generate three probabilistic outputs with associated confidence scores based on two arguments based to the inference script (run.py); name of model to be selected and path of a valid text file which contains the extracted chapter text on which the developer wants to run predictions. There are checks inside the code that ensures the script only runs if valid arguments are provided (see Figure 48). This inference script can only run with a GPU backend. For running predictions with the fine-tuned BigBird-Pegasus model, it’s necessary to have more than 7 GB free GPU otherwise it’ll give an ‘Out of Cuda Memory’ error. While the Figure 47, shows the run in a Google Colab cell, this same command can be executed in a server terminal. To run this command in a terminal just remove the exclamation mark (!) before the command. Before running this script ensure to change data_path variable in the code so that the model can get the checkpoint points from the appropriate folders. If the model name provided isn’t correct i.e. it’s not found in our list of model names; this script will default to predicting with the fine-tuned biobert model. The default model to run predictions can be modified by changing the model_name variable shown in Figure 48. If the text file path or type of training data i.e. trained on full ETD text vs trained on title and abstract information, provided are incorrect, this script cannot proceed and exits to save GPU resources and warn the developer early on itself as seen in Figure 49.
```python
if __name__ == '__main__':
    if len(sys.argv) != 4:
        print("Usage: python script.py <model_name> <txt_file_path> <type_training_data>")
    else:
        model_name, text_file_path, type_training_data = sys.argv[1], sys.argv[2], sys.argv[3]
        model_name_list = ['biobert', 'scibert', 'gpt2', 'bigbird']
        model_name = model_name.lower()
        if model_name not in model_name_list:
            print("Model Not Found!")
            print("Model Options:
        biobert
        scibert
        gpt2
        bigbird")
            model_name = "biobert"
            print("Defaulting to biobert ...")
        if os.path.exists(text_file_path):
            print("Text File Path is Valid, Proceeding ...")
        else:
            print("Text file path is not valid. Exiting ...")
            sys.exit(-1)
        if type_training_data not in ['Full_Text', 'Not_Full_Text']:
            print("... Invalid Type of Data Model was Trained on ...")
            print("Model Trained on Full_Text --> Full_Text")
            print("Model not Trained on Full_Text --> Not_Full_Text")
            sys.exit(-1)

main(model_name, text_file_path, type_training_data)
```

Figure 49: Checking Validity of Input Arguments.
9.0 Lessons Learned

9.1 Timeline/ Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Content</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of Jan</td>
<td>Finalize the project -&gt; Classifying ETDs launched, meet with the client, and understand the project.</td>
<td>Completed</td>
</tr>
<tr>
<td>By Feb 15</td>
<td>Literature Review (Metadata preprocessing, Manual Data Annotation and Data Quality Review with Client)</td>
<td>Completed</td>
</tr>
<tr>
<td>By Feb 28</td>
<td>Start implementing and generating baseline results for Text Classification Models. Review the existing website and identify the modifications needed to be accomplished.</td>
<td>Completed</td>
</tr>
<tr>
<td>By March 15</td>
<td>Prototype website designs using wireframes with our modifications. Sample the Text Classification Models and finalize one.</td>
<td>Completed</td>
</tr>
<tr>
<td>By March 31 (Iteration 1)</td>
<td>Dataset Integration and finalize website features.</td>
<td>Completed</td>
</tr>
<tr>
<td>By April 15 (Iteration 2)</td>
<td>Fine-tuning the model on document text and updating the website.</td>
<td>Completed</td>
</tr>
<tr>
<td>By April 30 (Final Iteration)</td>
<td>Finish integrating the model with the website.</td>
<td>Futureworks</td>
</tr>
</tbody>
</table>

Table 14: Timetable
9.2 Problems

9.2.1 NPM conflicting peer dependency error

The versions of node and npm on our local machines were different from the ones that were in the package.json. There were many errors when we tried to do “npm start”, especially with using React scripts. Initially, we were trying to first run the code base that the client provided for us and then make modifications based on the requirements such as removing pages that aren’t needed and adding features. We installed node.js and npm on our local machines but when we do “npm start” on the command line, we get the error as shown in Figure 50.

![Figure 50](image)

Figure 50: react-scripts command not found error when trying to run npm start

After we got this error, we tried to install react-scripts. Then, we did npm start again, and it was throwing this initialization error, as shown in Figure 51.
Figure 51: Initialization error with React when trying to do npm start

```javascript
error: unsupported
at new Hash (node:internal/crypto/hash:71:19)
      at Module._initModuleHash (node:crypto:333:18)
      at NormalModule._initBuildHash (/Users/vaishaliramesh/Documents/Capstone/Presentation/cs-5604-frontend-main/node_modules/webpack/lib/NormalModule.js:40:18)
      at /Users/vaishaliramesh/Documents/Capstone/Presentation/cs-5604-frontend-main/node_modules/loader-runner/lib/LoaderRunner.js:33:18
      at /Users/vaishaliramesh/Documents/Capstone/Presentation/cs-5604-frontend-main/node_modules/loader-runner/lib/LoaderRunner.js:33:18
      at LoaderRunner.run (node:internal/child_process/trace.js:190:12)
      at context.oncontextify (node:internal/child_process/trace.js:192:12)
      at Object.onOpenError (node:internal/child_process/trace.js:236:15)
      at processTicksAndRejections (node:internal/process/task_queues:96:5)
```

Node.js v18.15.0

Figure 52: Error message refusing connection to our local host

When we tried to do localhost:3000 on the Chrome browser, it refused to connect and displayed the error message as shown in Figure 52.
9.2.2 pg-native error

As shown in Figure 53, some of our group members were getting pg-native errors even though we had installed pg admin. We did “npm install pg” and also “npm install pg-native” in the terminal once we had downloaded Postgres for the backend. Our group member tried to remove pg from her system and install it again with supporting packages, but that also didn’t work. The member was able to solve this issue after taking many days to try different attempts. The solution is described in Section 11.2.3.

9.2.3 Proxy error

This was a major error that paused the UI work for about a week and a half trying to get it fixed. Initially, we did our own research by trying out solutions found online such as changing localhost to 127.0.0.1, trying to do “npm install --save http-proxy-middleware”, and checking to make sure the backend server is running, checked firewall and network settings. We tried to resolve the proxy error that is shown in Figure 54. We went to the TA and spent many hours trying different methods such as adjusting proxy server settings and creating our own rule for the inbound ports.
We tried sudo to see if it would run on an administration account with more privileges. We also tried to clone the Git repository and try it on different administration accounts and operating systems like Windows and Mac. The Mac user of our group didn’t get this error but got a different one (see Figure 55) when trying to run “npm start”.
9.2.4 Invalid options object error

Whenever we were doing “npm start” I kept on getting this error for the website. Initially, we have never got this error and were able to run the website by doing “npm start” but it would give a server error or a backend connection error. Once we got the database connection set up completely, this error was always there when we tried npm start. We spent around a week trying to fix this error.

Once the user chooses a file and hits upload, this error message, as shown in Figure 55, is displayed on the website. From conducting research on the error, we discovered that there is an issue communicating with our database server. We also discovered that this can be caused by proxy issues (see Figure 56), server issues, or a request timed out when trying to connect. We did try possible solutions such as checking the DNS setting, firewall, and network configuration. By following some of
the troubleshooting steps, we got that problem to be resolved, but we got a similar error later (refer to Section 10.2.6)

9.2.5 Database Connection Failed Error

![Upload or Database Connection error after the ETD file is uploaded](image)

Even though we had our Postgres database set up accurately, we kept on getting Upload or Database connection failure. This error was the major error that made us reevaluate the project and restructure to doing it on a local machine. Since the file was not getting uploaded correctly, and it didn’t connect to the database, we stopped using React. Refer to Section 10.3.4 to see an explanation of the solution we came up with after facing many errors with React and database connection.

9.3 Solutions

9.3.1 Resolving Installation Issues

To solve the installation issue, we added lines of code as shown in Figure 50, in the package.json. React was also giving a lot of errors, so it is important to make sure the versions of Node.js and npm are correct. To check, do “node -v” and “npm -v” on the terminal. The Node version we have is 18.15.0 and the NPM version is 9.5.0.

```json
"scripts": {
  "start": "react-scripts --openssl-legacy-provider start",
  "build": "react-scripts --openssl-legacy-provider build",
  "test": "react-scripts test",
  "eject": "react-scripts eject",
  "format": "npx lint-staged",
  "prepare": "husky install"
}
```

![Added lines of code in the package.json file to fix some of the initial error with using react-scripts](image)

To fix any npm errors, on the terminal inside the src folder, run the following command: “npm install react-scripts”. That should solve any errors that happen when trying to do “npm start”.
9.3.2 Fix for dependency error
To fix the NPM conflicting peer dependency error, we need to install and configure a few commands. The steps are as follows:
1. `npm install --save --legacy-peer-deps`
2. `npm config set legacy-peer-deps true`
3. `npm install react-google-login`
Refer to this website (Kentaroau, 2023) for more detailed steps and other solutions.

9.3.3 Fix for pg-native error
This error was because wrong versions were installed for pg and pg-native. We also needed multer and express to be installed and added to the dependencies in the package.json. Even though this solution fixed the error, there were still database connection errors when we were trying to do the front-end.

Figure 59: Dependencies in the package.json with the version number next to them.

We disregarded the old website code (code base that was given by the client) and redid the entire website with a new architecture using Node.js and Bootstrap. We could not solve the error in Section 10.2.5 because whenever we did “npm start”, that error was stopping us from running the file. A temporary solution we found was doing “node UploadETD.js” instead of doing “npm start” on the terminal, and that opened up the website. However, the website doesn’t look the same as the already existing discovery frontend (CS5604 Fall 2022 Class, 2022), so it didn’t have the style and beautification. This is why we decided to redo the architecture of the website and change from the requirements. The first requirement we had was to get the file that was uploaded to be saved on the local path in our local machine. Then, the file will be added to the uploads folder inside our website directory and then the database connection will be called. After, the input table in the Postgres database will be filled. The major requirements we had was that it needs to be fully functional in local and also be able to connect and modify the database. The new architecture (Node.js and Bootstrap) made it possible to complete those requirements. If we had initially started off with that rather than...
fixing errors we got from using react, we would have been able to do more work, such as running it on the server.

9.4 Model Limitations
For certain documents, the models tend to have an empty array of predictions instead of returning three probabilistic labels. However, the same document which works with one type of model doesn't work with another type of model. These errors are observed more in Non-BERT based models. In Figure 60 below, we show an example of the same wherein, the same chapter in txt form that is valid for the GPT-2 model fails for the BigBirg-Pegasus model during inference. The error seen below is because the entire tensor couldn’t be loaded into GPU but, when loaded it's empty the client was able to evaluate this using a server backend with 80GB GPU backend. Another limitation for text classification model development and fine-tuning script point of view is that due to time constraints, we were unable to adapt the .ipynb code book into a .py file that would run easily on server. Thus, with a powerful GPU connection and knowledge of ssh tunneling to connect server GPU to local and running the .ipynb code on local.

9.5 Future Work
While we set out with three modest goals in the beginning of this project, over the course of this semester, with our growing interest in the project, we tried to go above and beyond with the development of this project. To improve upon this work, future work for this project should focus on the following major aspects.

a) Developing a .py script to train larger models on a backend server instead of using the current .ipynb code book to train the models on advanced research computing (ARC) resources.

b) Using models with a larger maximum sequence length such as Longformer or Long-T5, and modifying the training code to allow mixed-precision training and accommodating gradient accumulation to use GPU resources more effectively. Focus on fine-tuning BLOOM BigScience model as it is well suited for this classification task (it’s resource heavy).

c) Find best hyperparameters for all models using grid search cross validation, Bayesian optimization, and automated hyperparameter tuning techniques to improve the performance of current models.
d) Building a better dataset by leveraging provided deep learning models classifying more ETDs and generating multiple labels for human evaluation through the IRB process.

e) Host the website on Server and Integrating with Inference Script called run.py
   i) The website needs to be hosted on the server in order to run the machine learning models. Also we need to take the input from the user (selecting the classification model from the experimenter page).
   ii) So that we can call an inference file called run.py to execute the machine learning model.

f) Running Predictions from End-User View:
   i) Once we get the input from the user. We need to provide the file path and model name to invoke the python script.
   ii) The output of the machine learning model should be displayed to the user in the experimenter page with the model they selected, predicted labels, and their respective confidence scores.

g) Testing the website on server.
   i) Make sure an user can successfully upload the ETD to the website and that it is classified correctly.
   ii) Test the integration of the frontend and the backend of the website by making sure the PDF information is added to the input table.
   iii) Ensuring that the trigger to the Machine learning model works correctly.

h) Integrating existing websites with the website developed in previous semesters and putting that in a Docker container with good documentation so that future researchers can build on it. Making the website more informative with dynamically visuals about the documents uploaded and the most commonly generated labels, etc.

i) Converting the renameDept() and mergeDept() methods from the 03162023_Analysis_ETD_Update.ipynb file into hash tables for easier maintenance and creating a lookup table to perform the STEM and Non-STEM mappings.
10.0 Appendix

10.1 Methodology
The methodology helps concretely identify the scope of the problems, conceptually organize the development efforts, and share and document progress with the client. (Based on Methodology assignment spec)

For details, see Figures 61 and 62, as well as Table 15.

10.1.1 Users intended to interact with our website

USERS:
➔ Curators:
   ◆ ETD Curation.
     ● Uploading ETDs to the website
➔ Researchers:
   ◆ ETD Categorization.
     ● Searches for ETD’s using keywords through the website using the document view page
➔ Experimenters:
   ◆ Assisting in assessing the model and creation of a gold standard dataset
     ● Display the results and store results to aid in creation of the gold standard dataset.
10.1.2 Tasks and Subtasks which make up the goal for each user

Personas - Curators, Researcher, Experimenters

User 1
- Curators - ETD Curation
  1A. Uploading ETDs to the website
  1B. Saves ETD pdf file path to the database and triggers text-extraction code on the pdf
  1C. Creates an ETD text file based on the pdf using the text-extraction code and saves it in the database

User 2
- Researchers - ETD Categorization
  2A. Searches for ETD’s using keywords through the website using the document view page
  2B. Utilizing the gold standard dataset developed, we show relevant documents based on the search query
  2C. Each ETD is classified into 3 ProQuest Labels using a text classification deep learning model with a confidence score for each label, which is used by the researcher to advance their work

User 3
- Experimenters - Assisting in Assessing the model and creation of a gold standard dataset
  3A. Uploading ETD Document and selecting required Model
  3B. Processing the document and generating the top 3 results with confidence scores
  3C. Display the results and store results to aid in creation of gold standard dataset

Figure 61: Tasks and Subtasks
10.1.3 Implementation process describing the services

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service Name</th>
<th>Input/ Input file name(s)</th>
<th>Input file fields (comma-sep)</th>
<th>Output/ Output file name</th>
<th>Output file ID</th>
<th>Libraries and Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>ETD upload</td>
<td>ETD.pdf</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Python, JS, Bootstrap, HTML</td>
</tr>
<tr>
<td>1B</td>
<td>Database Connectio n</td>
<td>ETD.pdf</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Postgres, Python</td>
</tr>
<tr>
<td>1C</td>
<td>Text-Extraction</td>
<td>ETD.pdf</td>
<td>N/A</td>
<td>ETD_clean Text.txt</td>
<td>N/A</td>
<td>Python, Postgres</td>
</tr>
<tr>
<td>2A</td>
<td>Search Query</td>
<td>Keyword from the search query</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>JS, Bootstrap, HTML, Postgres</td>
</tr>
<tr>
<td>2B</td>
<td>Data Retrieval</td>
<td>Keyword from the search query</td>
<td>N/A</td>
<td>List of matching data retrieved from database</td>
<td>N/A</td>
<td>JS, Bootstrap, HTML, Postgres</td>
</tr>
<tr>
<td></td>
<td>display retrieved data</td>
<td>keyword from the search query</td>
<td>n/a</td>
<td>displays search results dynamically</td>
<td>n/a</td>
<td>js, bootstrap, html, postgres</td>
</tr>
<tr>
<td>---</td>
<td>------------------------</td>
<td>-------------------------------</td>
<td>-----</td>
<td>------------------------------------</td>
<td>-----</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>2c</td>
<td>selecting classification model</td>
<td>model name</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>pytorch, js</td>
</tr>
<tr>
<td>3a</td>
<td>classification process</td>
<td>etd_cleanText.txt and model name</td>
<td>n/a</td>
<td>nested array with predicted label and confidence scores</td>
<td>n/a</td>
<td>pytorch, hugging face, colab</td>
</tr>
<tr>
<td>3b</td>
<td>displaying classification results</td>
<td>etd_cleanText.txt</td>
<td>n/a</td>
<td>displays the labels and confidence score dynamically (future works)</td>
<td>n/a</td>
<td>js, bootstrap, html, postgres</td>
</tr>
</tbody>
</table>
10.1.4 Workflow diagrams which cover each goal

Figure 62: Workflow

a. Workflow #1:

User → Goal 1 → Workflow 1

Workflow 1 = Service 1A + Service 1B + Service 1C

Service 1A: Uploading ETDs to the website

Service 1B: Saves ETD PDF file path to the database and triggers text-extraction code on the PDF

Service 1C: Creates an ETD text file based on the PDF using the text-extraction code and saves it in the database
b. Workflow #2

User → Goal 2 → Workflow 2

Workflow 2 = Service 2A + Service 2B + Service 2C

Service 2A: Searches for ETD's using keywords through the website using the document view page

Service 2B: Utilizing the gold standard dataset developed, we show relevant documents based on the search query.

Service 2C: Each ETD is classified into 3 ProQuest Labels using a text classification deep learning model with a confidence score for each Label, which is used by the researcher to advance their work.

c. Workflow #3

User → Goal 3 → Workflow 3

Workflow 3 = Service 3A + Service 3B + Service 3C

Service 3A: Uploading ETD Document and selecting required Model

Service 3B: Processing the document and generating the top 3 results with confidence scores

Service 3C: Display the results and store results to aid in creation of gold standard dataset.
11.0 Acknowledgements

We would like to thank our professor, Dr. Edward A. Fox for giving us the opportunity to pick this wonderful project, providing us with feedback on each step, and guiding us through the entire process. Additionally, we would like to thank our client, Ms. Bipasha Banerjee, for letting us work on this project and helping us develop the final outputs. We have given their contact information below:

- **Client: Ms. Bipasha Banerjee**
  bipashabanerjee@vt.edu

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


Hugging Face Community Contributors (2019). Hugging Face-Pretrained Models - transformers 2.4.0 documentation. Retrieved from https://huggingface.co/transformers/v2.4.0/pretrained_models.html


https://link.springer.com/chapter/10.1007/978-3-030-32381-3_16#Sec10

https://tomekkorbak.com/2022/10/10/compute-optimal-gpt2/


https://zenodo.org/record/883859#.ZF8OA3bMJPY

