Final Report
CS 5604: Information Storage and Retrieval

Electronic Thesis and Dissertation (ETD) Team:
Jiahui Fan, Sam Furman, Nicolas Hardy,
Javaid Manzoor, Alex Nguyen, Aarathi Raghuraman

December 16, 2020

Instructed by Professor Edward A. Fox

Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Abstract

The Fall 2020 CS 5604 (Information Storage and Retrieval) class, led by Dr. Edward Fox, is building an information retrieval and analysis system that supports electronic theses and dissertations, tweets, and webpages. We are the Electronic Thesis and Dissertation Collection Management (ETD) team. The Virginia Tech Library maintains a corpus of 19,779 theses and 14,691 dissertations within the VTechWorks system. Our task was to research this data, implement data preprocessing and extraction techniques, index the data using Elasticsearch, and use machine learning techniques for each ETD's classification. These efforts were made in concert with teams working to process other content areas, and build content agnostic infrastructure.

Prior work towards these tasks had been done in previous semesters of CS5604, and by students advised by Dr. Fox. That prior work serves as a foundation for our own work. Items of note were the works of Sampanna Kahu, Palakh Jude, and the Fall 2019 CS5604 CME team, which have been used either as part of our pipeline, or as the starting point for our work.

Our team divided the creation of an ETD IR system into five subtasks: verify metadata of past teams, ingest text, index using Elasticsearch, extract figures and tables, and classify full documents and chapters. Each member of the team was assigned to a role, and a timeline was created to keep everyone on track.

Text ingestion was done via ITCore and is accessible through an API. Our team did not perform any base metadata extraction since the Fall 2019 CS5604 CME had already done so, however we did still verify the quality of the data. Verification was done by hand and showed that most of the errors found in metadata from previous semesters were minor, but there were a few errors that could have lead to misclassification. However, since those major errors were few and far between, we decided that given the length of the project we could continue to use this metadata and added improved metadata extraction to our future goals. For figure and text extraction, we incorporated the work of Kahu. For classification, we first implemented the work of Jude, who has done previous work related to chapter classification. In addition, we created another classifier that is more accurate. Both methods are available as accessible APIs. The latter classifier is also available as a microservice. In addition, an Elasticsearch service was created as a single point of contact between the pipeline and Elasticsearch. It acts as the final part of the pipeline; the processing is complete when the document is indexed into Elasticsearch.

The final deliverable is a pipeline of containerized services that can be used to store and index ETDs. Staging of the pipeline was handled by the integration team using Airflow and a reasoner engine to control resource management and data flow. The entire pipeline is then accessible via a website created by the frontend team. Given that Airflow defines the application pipeline based on dependencies between services and our chapter extraction service failed to build due to MySQL dependencies, we were unable to deploy an end-to-end Airflow system. All other services have been unit tested on Git Runner, containerized, and deployed to cloud.cs.vt.edu following a CI/CD pipeline.

Future work includes expanding the available models for classification; expanding the available options for the extraction of text, figures, tables, and chapters; and adding more features that may be useful to researchers who would be interested in leveraging this pipeline. Another improvement would be to tackle some of the errors in metadata, such as that from previous teams.

Keywords: deep learning, natural language processing, ETD, electronic theses and dissertations, Elasticsearch indexing, metadata retrieval, classification

Acknowledgments

As a team we would like to first express our sincere gratitude and appreciation for Dr. Edward Fox and Bipasha Banerjee for providing us with great resources, always being readily available to help, and providing us with the PQDT data-set that we used to train our classifiers. We would also like to thank Palakh Jude for her help on this project. Her thesis code was useful for training our classifier and her advice and consultation were very helpful when we were building our first classifier. In addition we thank Sampanna Kahu for assisting with figure and table extraction, and providing consultation. Thank you to Prashant Chandrasekar and Rahul Agarwal for your guidance throughout the project, and thank you to the other project teams for help and advice with various parts of the project including Elasticsearch and the pipeline architecture. Finally, our team would also like to express our gratitude for the IMLS grant. This work was supported in part by IMLS grant LG-37-19-0078-19.

Contents

List of Tables 6

List of Figures 8

1 Overview 9

1.1 Introduction . . . 9
1.2 Project Management . . . 9
1.3 Problems & Solutions . . . 9
1.4 Future Directions . . . 11

2 Literature Review 13

2.1 Overview . . . 13
2.2 Data Processing and Extraction . . . 13
2.3 Overview of Natural Language Processing Pipeline . . . 14
2.3.1 Text Preprocessing . . . 14
2.3.2 Corpus Representation . . . 14
2.3.3 Clustering . . . 17
2.3.4 Classification . . . 18
2.4 Multi-label Classification . . . 18
2.4.1 One vs. Rest . . . 19
2.4.2 Binary Relevance . . . 19
2.4.3 Classifier Chains . . . 19
2.4.4 Label Powerset . . . 19
2.4.5 Adaptive Algorithm . . . 19
2.5 Machine Learning Classifiers . . . 19
2.5.1 Logistic Regression . . . 20
2.5.2 Support Vector Machine . . . 20
2.5.3 Random Forest . . . 20
2.6 Deep Learning . . . 20
2.6.1 Overview . . . 20
2.6.2 Recurrent Neural Networks . . . 21
2.6.3 LSTM Networks . . . 21

3 Requirements 24

3.1 Overall Project Requirements . . . 24
3.2 ETD Team Requirements . . . 24

4 Design 25

4.1 Data Extraction . . . 25
4.1.1 Tools for Extraction . . . 25
4.1.2 Metadata Extraction . . . 25
4.1.3 Fulltext Extraction . . . 26
4.1.4 Chapter Extraction . . . 27
4.2 Classification . . . 29

4.2.1 Categorical System . . . 29
4.2.2 Classification using Jude's Method . . . 30
4.2.3 Classification Using Probabilistic-Based ETD Classification . . . 35
4.3 ETD Metadata Fields . . . 38
4.4 ETD Processing Pipeline . . . 39
4.4.1 Validate Input . . . 39
4.4.2 Image Extraction . . . 39
4.4.3 Chapter Segmentation . . . 40
4.4.4 Text Extraction . . . 40
4.4.5 Classification . . . 40
4.4.6 Store and index metadata . . . 40

5 Timeline **42**

6 System Evaluation **44**
6.1 Chapter Extraction . . . 44
6.2 Classification . . . 46
6.2.1 Qualitative Evaluation . . . 46
6.2.2 Quantitative Evaluation . . . 48
6.3 Image Detection . . . 51
6.4 ETD Pipeline Evaluation . . . 52
6.4.1 Timing . . . 54
6.4.2 Pipeline Status . . . 54

7 User Manual **55**
7.1 Microservices . . . 55
7.1.1 Validation . . . 55
7.1.2 Figure and Table Extraction . . . 55
7.1.3 Chapter Extraction . . . 56
7.1.4 Fulltext Extraction . . . 56
7.1.5 Classification Using Probabilistic ETD Classification . . . 56
7.1.6 Ingestion . . . 56
7.2 APIs . . . 57
7.2.1 Chapter Extraction . . . 57
7.2.2 Classification Using Jude's Method . . . 57
7.2.3 Classification Using Probabilistic ETD Classification . . . 57
7.2.4 Elasticsearch Management . . . 58

8 Developer Manual **60**
8.1 Accessing ETDs . . . 60
8.2 Git Repositories for Services . . . 63
8.2.1 Microservices . . . 64
8.2.2 Chapter Extraction . . . 65
8.2.3 Figure Extraction . . . 68
8.2.4 PDF To Images . . . 68

8.2.5 Figure Inference . . . 68
8.2.6 Crop Images . . . 68
8.2.7 Text Extraction . . . 68
8.2.8 Classification . . . 68
8.3 CI/CD . . . 69
8.3.1 Unit Tests . . . 69
8.3.2 Gitlab Runner . . . 69
8.3.3 Deploy Tokens . . . 69
8.4 Airflow . . . 70
8.5 Adding Microservices . . . 74

Bibliography **78**

A Appendix **79**
A.1 Elasticsearch metadata mapping . . . 79
A.2 Sample goals . . . 83

List of Tables

1 Categories (based on the ProQuest Subject Category system) . . . 30
2 ETD Metadata Fields . . . 38
3 Tasks and Timeline . . . 42
4 ProQuest Test ETD Evaluation . . . 49
5 Postgres Metadata Schema . . . 61
6 Postgres Chapters Schema . . . 63
7 Airflow service descriptions . . . 71
8 Airflow goals . . . 72
9 Airflow reasoner table . . . 73

List of Figures

1 A simple CBOW model with only one word in the context [2] . . . 16
2 PV-DM model . . . 17
3 A flowchart of a simple decision tree . . . 21
4 Rolled and Unrolled RNN [21] . . . 22
5 LSTM's cell state [45, 16] . . . 22
6 A mismatch where the metadata record is missing the word "species" in the corresponding field . . . 26
7 A mismatch where the record has the more correct version of the title. For the purposes of searching, this is fine. . . . 26
8 Stages and Steps in ITCore's Workflow . . . 27
9 ETD Collections from VT, Penn State, and UIUC . . . 31
10 System Overview: Classification Pipeline [17] . . . 32
11 Doc2Vec embeddings generated by tuning hyper-parameters . . . 33

12 fastText Embedding Model parameters . . . 33
13 Label Powerset Models Generated . . . 34
14 System Overview: New Classification Pipeline . . . 35
15 TF-IDF Vectorizer params . . . 36
16 Logistic Regression Model Params . . . 36
17 Logistic Regression Model Predictions On A Statistics ETD . . . 37
18 Dividing and training ETD . . . 37
19 Pipeline for processing ETDs . . . 39
20 Example Segmented ETD. The left image is a snapshot of the the example dissertation's files. `Bart_AC_D_2017.pdf` is the original dissertation and `dublin_core.xml` contains its metadata. The right image depicts the resultant chapter PDF files. The chapter PDF files are named based on indices internal to ITCore and have no correlation to the page numbers of the original PDF. . . . 44
21 Dissertations Per Number of Extracted Chapters . . . 45
22 Dissertations Per Number of Extracted Chapters Between One and Ten . . . 46
23 Word cloud of abstracts . . . 47
24 Word cloud of titles . . . 47
25 Word cloud of abstracts . . . 47
26 Word cloud of titles . . . 47
27 Word cloud of subjects . . . 48
28 Word cloud of author departments . . . 48
29 Word cloud of subjects . . . 48
30 Word cloud of author departments . . . 48
31 Confusion Matrix . . . 49
32 Best Model Metrics . . . 50
33 Worst Model Metrics . . . 50
34 Second Pipeline Metrics . . . 51
35 Second Pipeline Parameters . . . 51
36 A case where the image detection microservice incorrectly labeled part of a figure . 52
37 Table of Contents labeled as an image . . . 53
38 Contents of a thesis folder . . . 60
39 Text Extraction service directory structure . . . 64
40 Structure of output directory upon running ITCore . . . 67
41 `.gitlab-ci.yml` file for the crop_image microservice . . . 70
42 Sample Goal ID 1 - `etd.pdf` . . . 83
43 Sample Goal ID 2 - Handle ID of dissertation . . . 84
44 Sample Goal ID 3 - Document Type ("D" for "Dissertation") . . . 84
45 Sample Goal ID 4 - ETD Metadata . . . 85
46 Sample Goal ID 5 - "Validate Finish" JSON created once the validation service has completed . . . 85
47 Sample Goal ID 6 - Directory of Images for each page in PDF . . . 86
48 Sample Goal ID 7 - (a) Bounding boxes overlaid for each Image in Goal ID 6 and (b) Bounding box value in text files . . . 87
49 Sample Goal ID 8 - Pictures extracted from the bounding boxes in Goal ID 7 . . . 88
50 Sample Goal ID 9 - 2 of 28 Chapter PDF files generated for Sample Goal ID 1 . . . 89

51 Sample Goal ID 10 - Text extracted from Sample Goal ID 1 90
52 Sample Goal ID 11 - Dublin Core XML 91
53 Sample Goal ID 12 - Directory of text extracted from Sample Goal ID 9 92
54 Sample Goal ID 13 - "Classify Finish" JSON created once classification is complete 92
55 Sample Goal ID 14 - "Pipeline Finish" JSON created once the last service has completed 92

1 Overview

1.1 Introduction

Twenty three years ago, Virginia Tech became the first university to require its students to submit their theses and dissertations electronically [43]. Over time, this corpus of research has grown to accommodate over 30,000 theses and dissertations. The accessibility of electronic theses and dissertations (ETDs) from both Virginia Tech and other institutions is crucial for improving the quality and efficiency of research. We hope to help the Virginia Tech ETDs live up to their potential via the extraction of key metadata attributes and the classification of ETD chapters. This report serves as a record of the problems we faced and solutions we developed to accomplish our data goals.

Our efforts with ETDs are made in concert with those of other students. Virginia Tech's class, CS 5604: Information Storage and Retrieval, is a collaborative effort to build a state-of-the-art information retrieval and analysis system. The system will contain data from three main collections: ETDs, web pages, and tweets. Users will be able to interact with each of these three content categories through a unified frontend.

1.2 Project Management

The nature of our prescribed tasks and the advice of our subject matter expert (SME) indicate that the creation of a minimally viable end-to-end pipeline with an extensible architecture is our key priority. In order to maximize the efficiency of our efforts, we aim to leverage existing technologies and draw upon previous approaches where possible. We have distributed our six team members to work on different parts of the end-to-end pipeline. The distribution of roles is as follows:

- Data Indexing: Nicolas Hardy, Jiahui Fan
- Data Ingestion: All members
- Data Extraction: Alex Nguyen, Aarathi Raghuraman, Samuel Furman
- Classification: Javaid Manzoor
- CI/CD: Nicolas Hardy

Communication occurred during class meetings and a weekly meeting independent of class time. We used Zoom for video calls and Discord for chat. Any artifacts that the team creates are stored on a shared Google Drive. Our code is tracked in a Git repository stored on Virginia Tech's Gitlab server. We employed common DevOps tools including Ally and continuous integration to keep our development on track and avoid regression.

1.3 Problems & Solutions

The Fall 2019 team working on ETDs, from the CME [20] and TML [28] groups, provided us a solid background on challenges and limitations they faced which helped us improve on their analysis from the start. Some of the challenges faced by the Fall 2019 team include:

- Figure, Table, and Formulae Extraction: The 2019 team was unable to find a reliable library to extract figures, tables, and formulae. In fact, the 2019 team was unable to extract any figures or tables. We initially experimented with PyMuPDF for figure extraction and pdf-plumber for table extraction, which gave us mediocre results, based on manual validation. Further some ETDs are scanned documents, which neither PyMuPDF nor pdfplumber can handle well. Given the 2019 team's experience and our results from using built-in libraries, we switched to use the latest research in figure and table extraction based on Kahu's thesis [19].

- Chapter Extraction: Past teams, including the Fall 2019 team, used GROBID for chapter extraction, and mentioned GROBID's quality of extraction to be poor. This was further validated by Jude's thesis on chapter classification [17]. She compares GROBID and ITCore, a Java rule-based chapter extraction tool originally built for extraction from textbooks. The number of chapters extracted by GROBID ranged from 1 to 313 with no inferable distribution, while ITCore ranged from 1 to 123 with a long-tailed normal distribution centered around 6-10 chapters. These results suggest that ITCore is more reliable for chapter extraction, and thus is our current solution.

- Full Text Extraction: The 2019 team used PyPDF2. We realized that PyPDF2 has not been updated since May 18, 2016. Additionally, there are several characters in a PDF that cannot be extracted by PyPDF2. We needed to find another tool to extract full text. PyMuPDF is a new tool to extract text from PDF files. This tool improved the results of extraction, with more text pulled from a PDF.

- Classification: The Fall 2019 team utilized Named Entity Recognition (NER) models to identify the person, organization, and location information in ETDs. Since we are more interested in identifying subjects for search-ability and not metadata extraction, we decided to use the currently available metadata along with the PDF to classify each ETD from context within the text.

While we navigated some of the past challenges, we ran into new challenges as mentioned below:

- Metadata Validation: There was no explicit information about the quality of the metadata (e.g., which records were malformed, what records were specifically related to ETDs), so we had to manually verify the records that were created by previous teams.

- Chapter Extraction: Even though ITCore performs better than some of the top performing PDF extraction tools, about 40% of our sample dataset was not processed by ITCore [1]. Also, ITCore requires ETDs as PDF files and cannot handle scanned documents.

- Figure and Table Extraction: Our initial efforts leveraged tools that depend on the innate structure of born-digital documents. Midway through the semester, we pivoted to using the work of Kahu et al. [19].

We also encountered significant challenges attempting to deploy our microservices to Apache Airflow. More details about these challenges and the final state of the pipeline are discussed in Section 6.

[1]Some of the documents could not be processed by ITCore because ITCore was not able to properly read the documents' table of contents

1.4 Future Directions

Given our current challenges, a few ideas for future directions include:

- Metadata Validation: Create an automated script that can check the format and accuracy of the metadata being extracted based on rules. Coming up with concrete rules will be the most challenging part of this task.

- Chapter Extraction: Many of the PDF files were skipped by ITCore due to poor table of contents (ToC) extraction. From Section 4.1.4, we know that the ToC is the foundation for ITCore's rules. From the Fall 2019 CME team [20] and Jude's work [17], we know that PyMuPDF is a great tool for ToC extraction. Future teams can leverage PyMuPDF to perform ToC extraction and feed the extracted ToC to ITCore. The challenge here will be that PyMuPDF is a Python tool, while ITCore is a Java tool that is extremely complex.

- Text Extraction: As mentioned in Section 1.3, many of the older dissertations and theses are scanned in as images. Performing any manipulation of the text within these images is a challenge, hence the need to extract the text to perform analysis. Mehul et al. [30] mentions many ways of using image segmentation for text extraction, which can serve as a guide for future work in this area.

- Figure Extraction: The current implementation of figure extraction is somewhat of a half measure. The PDF-to-image and crop-image services that bookend the inference step of the figure extraction pipeline are a temporary patch. The deepfigures-open pipeline constructed by Kahu and others already supports this functionality. Unfortunately, after wrestling with the full pipeline for a while, we were only able to integrate the core inference step into the production environment. Therefore, a full adaptation of the pipeline should be attempted in the future. In the absence of this full adaptation, we were also unable to provide services to train and evaluate new figure and table extraction models. This use case is likely to be incredibly useful to research-oriented users. Supporting training of new models comes with the additional hurdle of incorporating GPU resources.

- Classification: Within our current classification pipeline there are various potential improvements possible. Implementations of zone weighting could potentially help weight different zones such as title and keywords highly. Rigorous hyper-parameter testing can help fine-tune parameters. Another possible improvement would be to change the baseline model and compare it to our current Logistic Regression baseline model. According to [27], Naive Bayes works great for text classification.

- Containerize Classifiers: This is a cross-team note. Explicitly understanding Airflow's input requirements and curating inputs based on each collection would be helpful. Because of the current Airflow integration requirements our pre-trained classifiers are not containerized. The classification service has to continually reload the classifiers which is inefficient. A possible solution could be to host the model on another, persistent, container and communicate with that container to classify.

- Supporting Multiple Collections: Currently, every ETD that goes through the pipeline gets indexed into Elasticsearch under 'etd'. To support more indices, only the etd_ingestion

service would have to be modified. A new goal would have to be added in order to specify the name of the ETD index. The Python Elasticsearch API (https://elasticsearch-py.readthedocs.io/en/master/) has a robust set of functions that can be used to manage the logic behind sending a record to the correct index. Another extension is adding support for visibility settings on collections (e.g., making a collection private to a specific user or group of users). Supporting visibility settings would require modifications to multiple microservices, the Postgres database where the metadata is stored, and the frontend created by the FE Team.

- Full GitLab Pipeline: Currently, we use the Gitlab Runner to test the microservices in isolation. We had hopes of having the ability to test the full pipeline in a single series of stages in the Gitlab Runner, however the design of the runner prevents this from being straightforward. Specifically, the problem with trying to test the entire pipeline at once is that we have to keep a Postgres database persistent across stages. The Gitlab Runner executes each part of a test pipeline in isolation. Any containers (e.g., Postgres and Elasticsearch instances) that are used for testing are reset in each stage of the pipeline to maintain this isolation. As a result, maintaining a single database for the duration of a pipeline is impossible. One solution could be to have the pipeline interact with a persistent database that is outside the control of the Gitlab Runner, however this would require additional work to ensure that concurrent pipelines could not interfere with each other when utilizing said database. Another solution to this problem could be to fetch the intermittent data from the database at the end of every stage and store that as an artifact. This would allow the subsequent stage to repopulate the new Postgres instance, however this might be more trouble than it's worth.

- Document Identifier: Currently, all documents in our system are identified by the handle number. In future endeavors each document should be identified by their full URI.

Lastly, possible future work could include expanding the capabilities for each of our services, specifically offering more tools for researchers who want to use this workflow on their own collections.

2 Literature Review

2.1 Overview

The University Libraries have been working towards digitising all ETDs and uploading them on the VTechWorks system. The system currently hosts at least 19,000 Masters theses and 14,000 Doctoral dissertations. Most of the documents are in PDF form, but some of the older ones are scanned. In service of our data goals, we aim to extract key metadata, definitions, and figures from these electronic documents. The processing of PDF documents that vary in format and structure has proven a difficult task. Fortunately, we have access to a large collection of previous efforts from which we can draw useful techniques and approaches. Upon completion of data extraction, we hope to add value via classification of ETD chapters. In this section we explore existing works that tackle data extraction and classification in the context of electronic documents.

2.2 Data Processing and Extraction

Reports on previous efforts in CS5604 have valuable information about which processes and techniques bore fruit and which became dead ends. GROBID (GeneRation of BIbliographic Data), ScienceParse, and a number of Python libraries including PDFMiner and PyPDF2 are the most common set of tools utilized for metadata extraction [3, 6, 20]. It is important to note that previous works found that no single tool functioned as a panacea for data extraction tasks. To illustrate, despite finding some success with both GROBID and ScienceParse, the 2019 team indicated that both tools occasionally mangled the content of documents [20]. Despite eventually using GROBID for its ease of configuration, the 2019 team cautioned that metadata extraction remains a challenge for the tool. They also attempted chapter-level text extraction on GROBID's output Text Encoding Initiative (TEI) document.

TEI is an application of XML and was developed to encode machine-readable texts. The 2019 team used XPath, a query language for selecting nodes from an XML document, to extract the chapter-level text, but had poor results with subsections being segmented into chapters [20]. Research by Jude showed that Intelligent Textbooks Core (ITCore) performed better than GROBID for chapter-level text extraction [17].

Some other teams have experimented with extracting figures from ETDs. The most common attempts range from using libraries such as GROBID and PyPDF2, to training models for extracting information via libraries like PDFFigures2.0 and Deep Figures [6, 8, 37]. These works indicate that GROBID has the ability to identify figures with a reasonable degree of accuracy. PyMuPDF and pdfplumber represent more recent efforts on providing PDF processing utilities in Python [?] [29]. Both of these tools leverage the underlying document structure of born-digital ETDs. As a result, they are ill-equipped to deal with scanned ETDs.

Previous teams employed the Elasticsearch search engine to add scalable search capabilities to ETDs using extracted data [10]. Using Elasticsearch introduces an additional criteria upon which extraction tools must be measured. The format of the extraction output must be amenable to ingestion into Elasticsearch. The 2019 team indicates that it is possible to convert GROBID's TEI documents into the subset of JSON that Elasticsearch accepts via Python scripts [20]. Beyond format, it is important to consider the content of extracted data when using Elasticsearch. The 2019 Elasticsearch team determined that metadata fields like author, date, title, keywords, etc.

were important for getting the most out of the search engine. Their work can function as a detailed foundation upon which we can set up Elasticsearch for data ingestion [24].

From these previous works we note that there has been much success towards the goal of indexing ETDs, but there are still complications that hurt the quality of the resultant content. The work presented in this report aims to extend the functionality and increase the quality of the works of previous teams. Section 4 contains a more comprehensive comparison of available extraction technologies.

2.3 Overview of Natural Language Processing Pipeline

Natural Language Processing (NLP) models help classify/cluster bodies of text by understanding the underlying context. NLP models must be trained with sets of text and/or expected categories. Before we can process any of our ETDs with NLP models we must convert the text into machine understandable format. Various conversion methods are explained below, but before we can convert the text using the techniques we must preprocess the text. [2].

2.3.1 Text Preprocessing

Text Preprocessing is a traditionally important step for Natural Language Processing tasks. Depending on the task, there are various preprocessing techniques we can use. Preprocessing can be helpful to reduce the corpus size and remove words that provide little to no context among many others. This helps boost NLP efficiency. [2]:

- **Punctuation Removal**: Remove punctuation, accents, and diacritics to avoid textually irrelevant elements. This should be applied carefully depending on the underlying language to which preprocessing is applied. Removing these elements from documents in English has little to no effect within a classification task. Removing punctuation also helps with removing textually irrelevant elements [27].

- **Stop Word Removal**: Remove frequent words that add little to no context to the text (e.g., "the", "a", "this", "is", "will"). This helps reduce corpus size but it should be applied carefully if phrases hold importance [27].

- **Case Folding**: Convert all words into the same case. This can help conflate terms like "Automobile" and "automobile." Case-folding text with lower case is a popular method that works great in general. [27]

- **Stemming and Lemmatization**: The goal with either of these techniques is to reduce inflectional forms of words or convert the words to its base forms. This helps normalize words like organize, organizes, and organizing to a singular form. This can effectively reduce the size of the corpus vocabulary. [27]

2.3.2 Corpus Representation

In this section we will discuss a few techniques that can be used to represent text in machine understandable format.

Term Frequency — Inverse Document Frequency Term Frequency — Inverse Document Frequency (TF-IDF) is a technique from information retrieval to build statistics on a corpus to reflect how important a word is within it. It first generates each word's **term frequency (TF)** within each document.

$$\textbf{TF(term, doc)} = \textbf{count}(term, doc)/\textbf{vocab}(doc) \tag{1}$$

Also, a word's inverse document frequency must be generated. **Inverse document frequency (IDF)** measures the importance of a word within the entire corpus. A word that appears more often then other words has a lower score than a word that has fewer occurrences. The rarer a word within the corpus, the higher its score. However, words that occur a very small number of times may indicate spelling mistakes or other types of out-of-vocabulary (OOV) terms within a corpus.

$$\textbf{IDF(term)} = log(\textbf{No. of Documents}/\textbf{count}(term, corpus)) \tag{2}$$

IDF helps weight contextually less relevant words like stop-words with a lower score, and more relevant words – like "neural" – higher.TF-IDF is the product of the two scores.

$$\textbf{TF-IDF(t, doc)} = \textbf{TF}(term, doc) * \textbf{IDF}(term) \tag{3}$$

TF-IDF mathematically reflects the importance of each term in a document in a corpus.

Embedding Words must be converted into a continuous vector representation for some machine learning models to understand them. Embedding is the mapping of discrete categorical variables into continuous vectors. Ideally, the goal of embedding is to create some sort of context within the semantics and vocabulary by placing entities with similar semantics closer. It does this using different techniques, such as neural networks to map words to vectors. This means that words that are similar are grouped closer. Words like mangoes, apples, and bananas would be closer, and words like apples and cars would be further away [2].

Word2Vec Word2Vec is the first word embedding model proposed, and one of the more popular ones. The embedding relies on shallow neural networks to generate word context. A shallow neural network only contains one hidden layer compared to a deep neural network that can have multiple hidden layers. Word2Vec starts by converting the text vocabulary into a series of one-hot encoded vectors – one for each word – representing the position of each word. The goal here is to find close spatial positions between each word to build context [31]. Word2Vec relies on two different approaches which both use Neural Networks, namely the common bag of words (CBOW) method and the Skip Gram method. Figure 1 is a high level overview of the CBOW method.

The CBOW and Skip Gram methods are interestingly related. CBOW relies on finding a target word using a sentence of related words. It builds off the one-hot encoded vector and finds the target word off of positioning. The Skip Gram method is basically the reverse of CBOW. The Skip Gram method takes an input term and tries to find a collection of terms that are commonly used with the input term. Word2Vec works great overall, but does not work well with rare words or words not present in the out-of-vocabulary (OOV) dictionary.

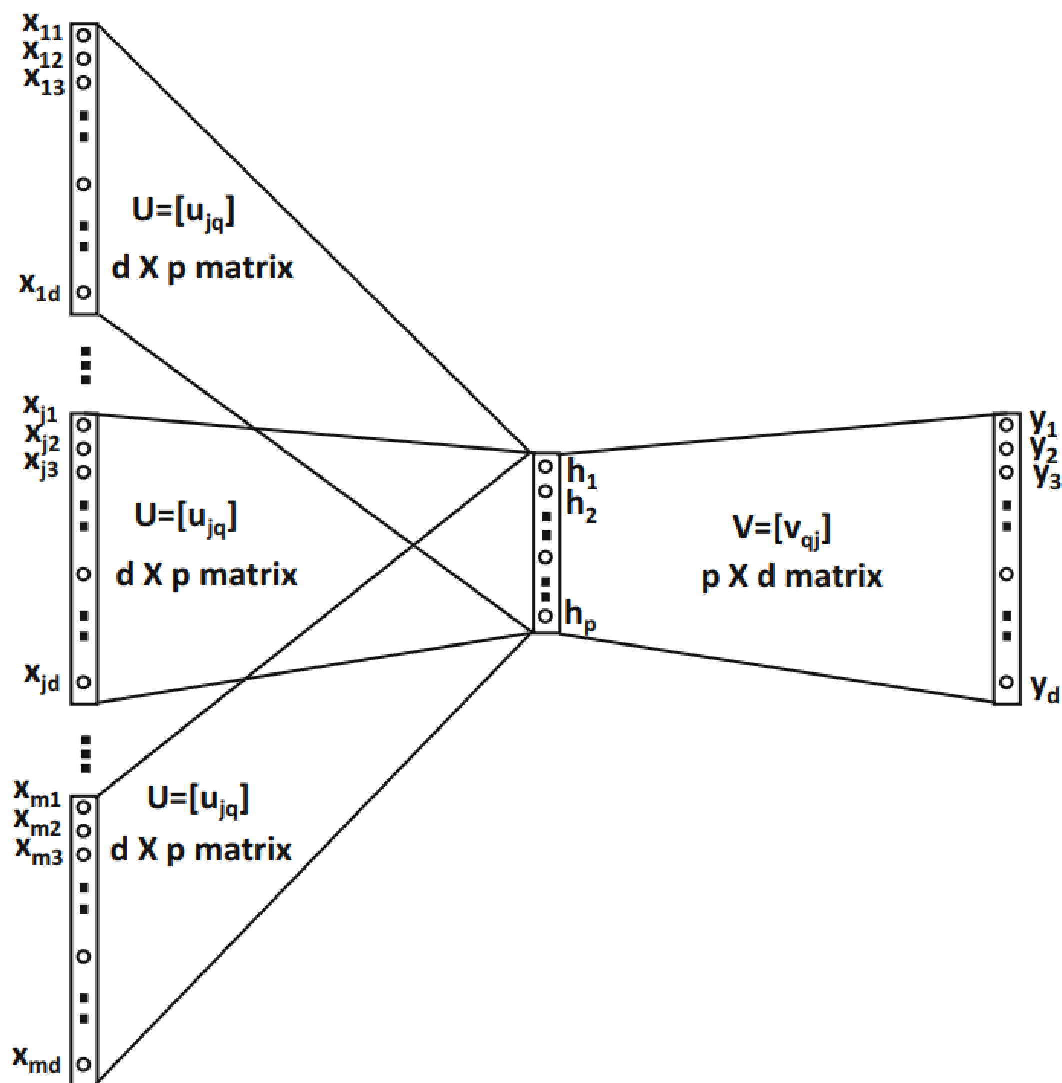


Figure 1: A simple CBOW model with only one word in the context [2]

fastText The fastText embedding process was created by Facebook's AI Research Lab (FAIR) [11]. Compared to the other embedding techniques, fastText differs because it generates vectors based on an n-gram character scheme instead of generating word vectors. For example, the word "sunny" with a 3-gram scheme would be divided into <su, sun, unn, nny, ny>. Since the words get divided, fastText works great with rare words or even words not present in the OOV dictionary. The neural network architecture mentioned in Section 2.6 is similar to the CBOW model.

Doc2Vec As the name suggests, Doc2Vec is built upon the Word2Vec embedding. While the Word2Vec embedding works fine with shorter text, it does not work well with long text such as theses and dissertations. Doc2Vec attempts to solve this problem by creating a numeric representation of document segments, e.g., those represented by a paragraph-id. These segments are

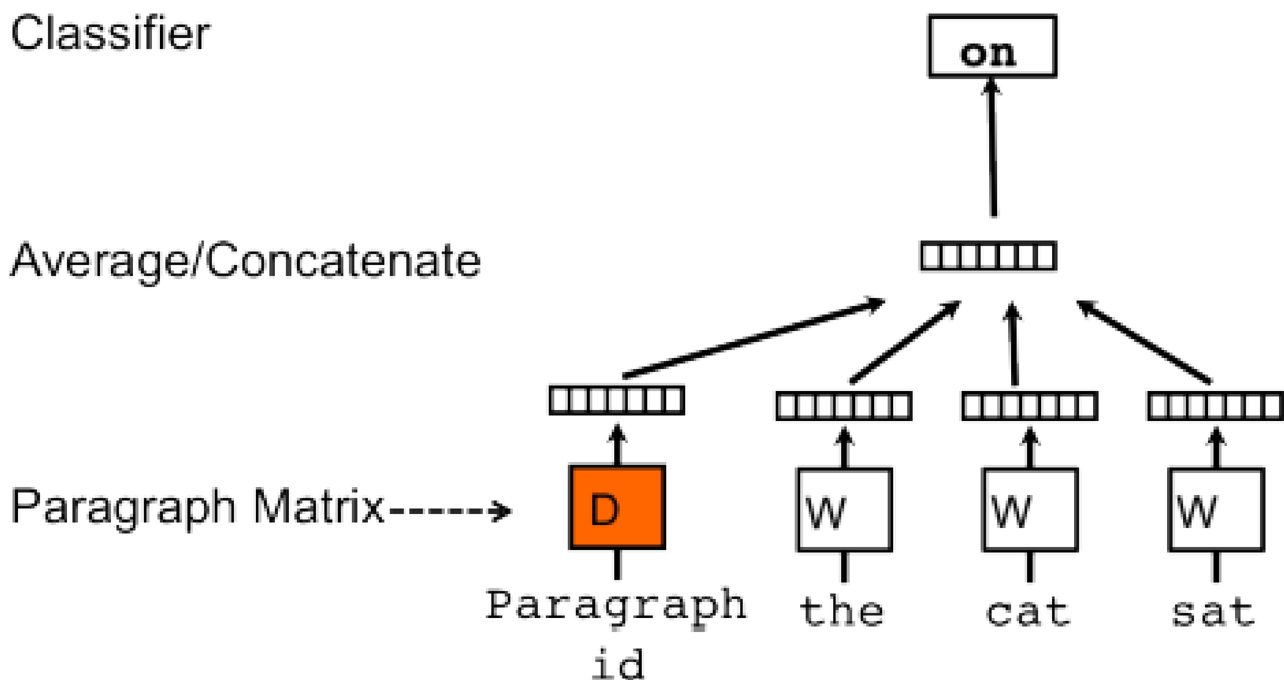


Figure 2: PV-DM model

decided by heuristic rules. It could be paragraph, chapters, etc. [23].

Figure 2 shows the Distributed Memory Version of the Paragraph Vector model (PV-DM) method Doc2Vec uses. The PV-DM method is similar to Word2Vec's CBOW method, but adds a paragraph-id to the sliding window of word. This helps Doc2Vec build word context within long text.

2.3.3 Clustering

Clustering is an unsupervised method; that is, one does not know which cluster each document belongs to. In order to group documents together, one can either assign a document to belong to a single cluster (hard clustering) or assign a probability to a document belonging to each cluster (soft clustering). While some ETDs have labels available in the form of their metadata, many ETD chapters lack any labels. Luckily, clustering can still function in the absence of these labels. Within clustering there are a variety of approaches commonly used for text analytics:

- **Generative Models**: Theseassume that the document-term matrix is generated from a probabilistic process. Within generative models, deterministic and probabilistic matrix factorization methods assume that the complete matrix is generated by a probabilistic process. In contrast, probabilistic mixture models assume only the rows of the document-term matrix are generated from probabilistic processes. Once the distribution of the underlying generative process is understood, the data is clustered based on which distribution it falls under.

While the creation of this technique is recent, there have been a few different enhancements in this area for NLP [41, 39].

- **Similarity-Based Algorithms**: All similarity-based algorithms cluster based on some form of a distance matrix that quantifies the distance between or similarity of two objects. In our case, these objects correspond to complete ETDs or ETD chapters. Similarity-based algorithms are some of the most mature clustering methods. They are easy to understand and interpret, but can be high in space and computational complexity. Commonly used similarity-based algorithms include k-means [4, 44, 9] and hierarchical clustering [12, 25].

- **Advanced Methods**: Graph partitioning methods and ensemble methods add an additional layer of complexity and accuracy [7].

2.3.4 Classification

Unlike clustering, classification relies on existing labels. As mentioned previously, not all the ETDs and chapters have keywords/labels associated with them. Therefore, the utility of classification methods is contingent upon our ability to assign meaningful labels. Labels can be obtained through extraction, such as by using the tf-idf frequency matrix to pull the top k keywords for each document. Alternatively, we can manually access and quantify keywords for some of the documents (semi-supervised). Additionally, advanced methods such as deep learning can learn from associations in existing classifications. Jude [17] presents a dichotomy of classification algorithms:

- **Machine Learning Classifiers**: These include tree-based classifiers such as Decision Trees and Random Forests, Naive Bayes Classifier, Linear Classifiers such as logistic regression, and non-parametric classifiers such as k-Nearest Neighbors (kNN) and Support Vector Machines (SVM).

- **Deep Learning Classifiers**: Deep learning is a popular topic given its success in various fields including NLP. Some of the most common architectures used in NLP include: Recurrent Neural Networks (RNN) such as Long-Short Term Memory Networks (LSTM), Convolutional Neural Networks (CNN), Deep Belief Networks (DBN), and Hierarchical Attention Networks (HAN). These architectures can be combined to compliment the strengths of each architecture as well as reduce the deficiencies [22]. As mentioned in a 2020 survey of deep learning methods for NLP, it is believed that these methods are learning the template and syntactic pattern of text instead of the logic or inference of the text [32]. Deep learning is a growing field with a lot of promise for applications in NLP.

In the next sections we will talk about the nuances of multi-label classification, and elaborate more on some of these classifiers.

2.4 Multi-label Classification

Within an ETD, chapters can contain work from different disciplines. Since these classifications are not mutually exclusive, there is bound to be overlap. This makes our classification problem a multi-label classification problem where each chapter can have multiple classes, and we will have to choose the top few classes to represent it. Traditional classification models are built on

single label classification. This leads us to divide our multi-label problem into multiple single-label classification problems using common machine learning techniques.

2.4.1 One vs. Rest

One vs. Rest is a traditional two-class classification technique. We build multiple independent classifiers that represent a single class vs. the rest. For instance, if our classes were [red, green, blue] we would build three classifiers, one for each class. For red, it would build a classifier testing for [red, [green, blue]]. The classifiers with the highest confidence are then picked. The problem with the One vs. Rest method, however, is that it does not take class correlation into account [27].

2.4.2 Binary Relevance

This is similar to the One vs. Rest method as it generates multiple independent classifiers for each class, but then tests for a [yes, no] prediction whether the classifier qualifies. It next glues together the predicted outputs as the multi-label output. Just like the One vs. Rest method, it does not take into account correlation. The method is popular because of how easy it is to implement. [27]

2.4.3 Classifier Chains

Classifier chains are built upon multiple independent binary classifiers chained together as $C_0, C_1, ..., C_n$ where C_i takes into account the prediction of all of the classifiers before it. This process helps maintain any class correlation. However, it is computationally more complex then the previous discussed methods. The number of classifiers to generate usually matches the number of classes. [34].

2.4.4 Label Powerset

The label powerset method works by transforming a multi-label problem into a multi-class problem where one class represents all unique label combinations. Each class has its own independent classifier. This classification technique retains class correlation as well. However, the label powerset can be computationally expensive if there are too many classes and combinations to represent. This leads to a deterioration in performance, especially given how diverse the catalog of classes within ETDs can be [42].

2.4.5 Adaptive Algorithm

This algorithm works by adapting a single label classification system to a multi-label classification system by changing decisions in the cost/decision functions. A common adaptive algorithm is the multi-label lazy learning approach, ML-KNN, which is derived from the K-Nearest Neighbour algorithm. This method also has a high computational complexity, but it does take class correlation into account [46].

2.5 Machine Learning Classifiers

In this section we describe the different machine learning classifiers currently in use. These are "traditionally" single-label classifiers that we extend using our multi-label techniques as explained

above.

2.5.1 Logistic Regression

Logistic regression is a regression technique used to classify a dependent variable based off one or more independent variables. At its core, it relies on the sigmoid function to normalize the function to a value between 0 and 1 inclusively. Since the returned value is squeezed between 0 and 1, we can treat the returning value as a probability for each class. With our multi-label problem, it can allow us to rank the highest to lowest probabilities generated by each class [2].

2.5.2 Support Vector Machine

Support Vector Machines, or SVMs, is another classic machine learning classifier. After mapping all of the independent input points, SVMs attempt to find an N dimensional plane, matching the N features, to separate the data points. These "vectors" are built such that each associated cluster is equidistant, thus helping identify different classes within the data [40]. The SVM vectors try to find the optimal "hyperplane" whose dimensions match the number of features, and also separate the classes by being equidistant from each other.

2.5.3 Random Forest

The Random Forest classifier builds upon another traditional machine learning technique, decision trees. The underlying concept is used by everyone in their daily lives. It is a supervised learning algorithm that decides the target variable's category based on a chain of decisions made by considering the feature columns in the data. It can be visualized as a flowchart of decisions leading to an answer [15].

The Random Forest classifier builds upon decision trees by instantiating an ensemble of trees that randomly selects the data to train its ensembles on. It is completely random and leaves less room for bias compared to decision trees. It relies on the central value theorem, that a statistic of many smaller samples within a population will center around the true statistic of the population.

2.6 Deep Learning

2.6.1 Overview

Given the amount of data gathered each year, researchers are constantly trying to find new ways to make sense of this data using machine learning techniques. Here, we take a dive into a sub-category of machine learning called deep learning. It is based upon Neural Networks (NNs) that consist of neurons organised in layers. These layers are connected to each other based on weights, and each neuron has its own bias. One layer's activation influences the activation in the next layer, and so on. There is an input layer and an output layer with one or more hidden layers in between. The "learning" signifies the ability of deep learning models to modulate their weights based on new information. This is done by a process known as back-propagation [2].

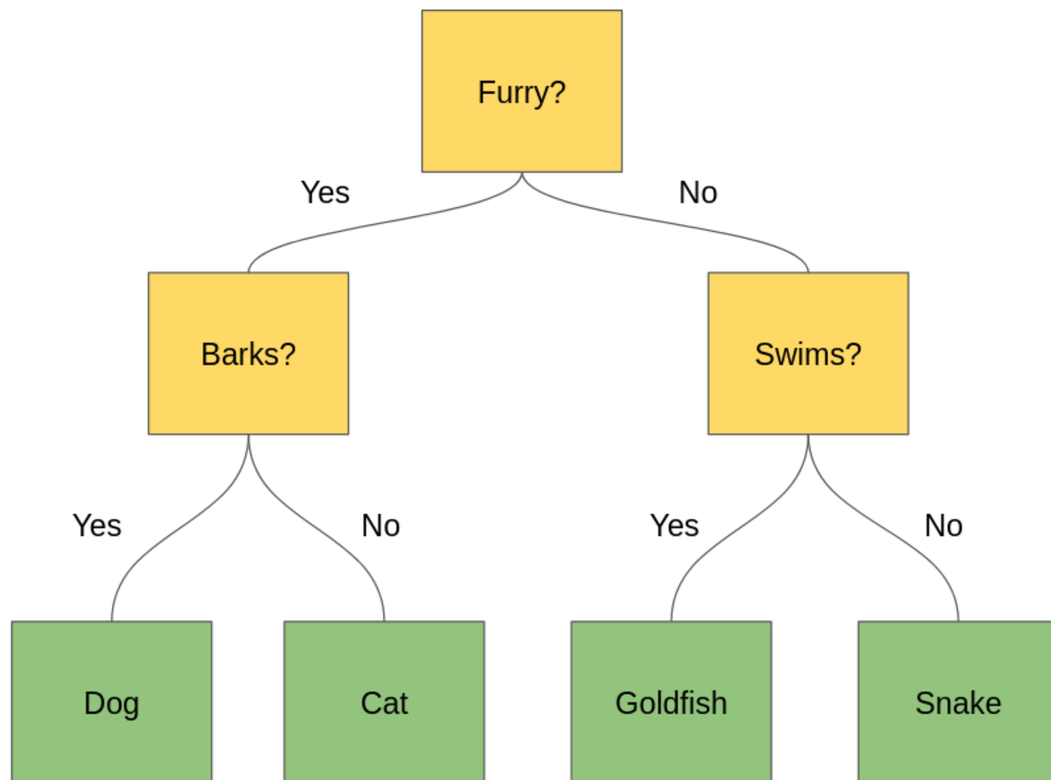


Figure 3: A flowchart of a simple decision tree

2.6.2 Recurrent Neural Networks

Within deep learning, there are special models known as Recurrent Neural Networks (RNNs) that have led to muchresearch in recent years. These RNNs are special because unlike NNs, RNNs use sequential information. This is one of the reasons why RNNs are so popular for natural language processing problems [1]. In essence RNNs have "memory" that makes them well suited to processing connected pieces of information like text and speech.

Figure 4 depicts the unrolled structure of a RNN. It divides the input in order and retains information of inputs sequentially. In this figure, h retains the weights and biases of the network and W refers to information that is being transferred between each input. An important point to distinguish, however, in the unrolled picture is h. The h block is always the same, and it keeps feeding itself information based on the last output [21]. RNNs are ill suited to tracking relationships over long periods of time. This is a well studied issue known as the vanishing gradient problem. This problem deserves attention in the context of classifying long works like ETDs.

2.6.3 LSTM Networks

The Long Short-Term Memory (LSTM) is an improved algorithm in response to the deficiencies of RNNs. Traditional RNNs have trouble carrying information between longer sequences. LSTMs

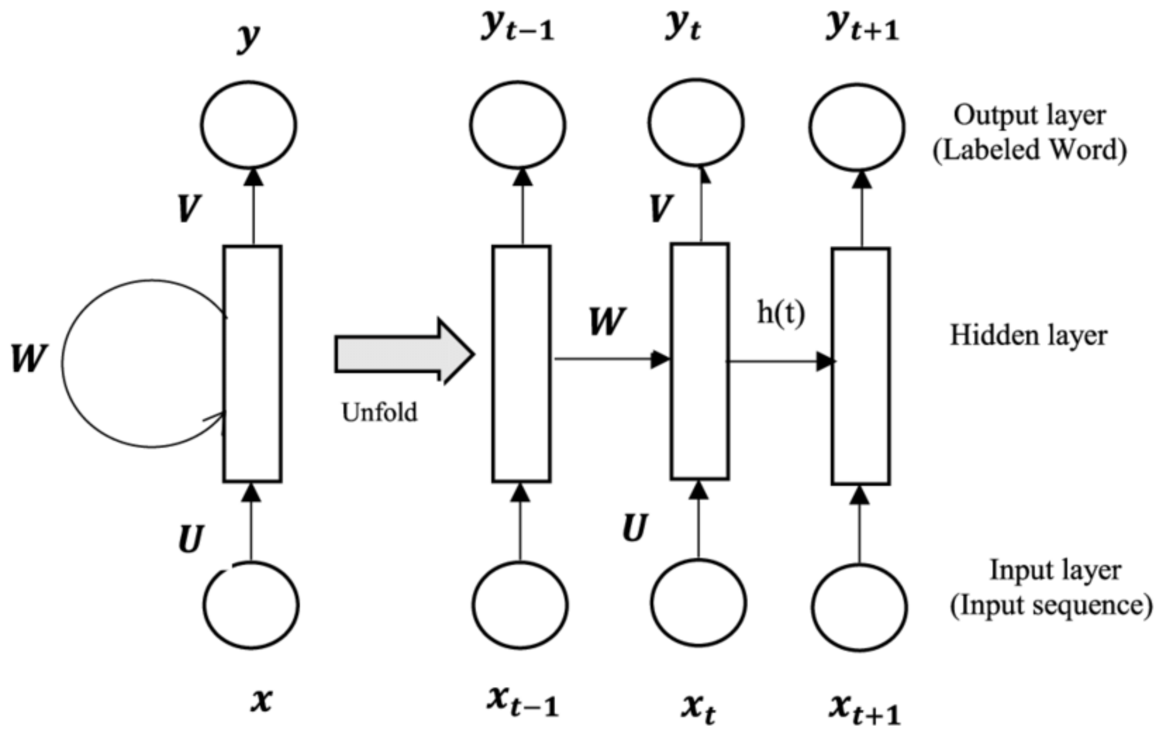


Figure 4: Rolled and Unrolled RNN [21]

include a "memory cell" that retains long term information. This cell can be accessed through a gate. Each cell has a constant monitoring its error rate. The gates between two cells learn to open and close based on the error within each memory cell.

Figure 5 depicts a typical LSTM memory cell. Compared to a traditional RNN, an LSTM cell

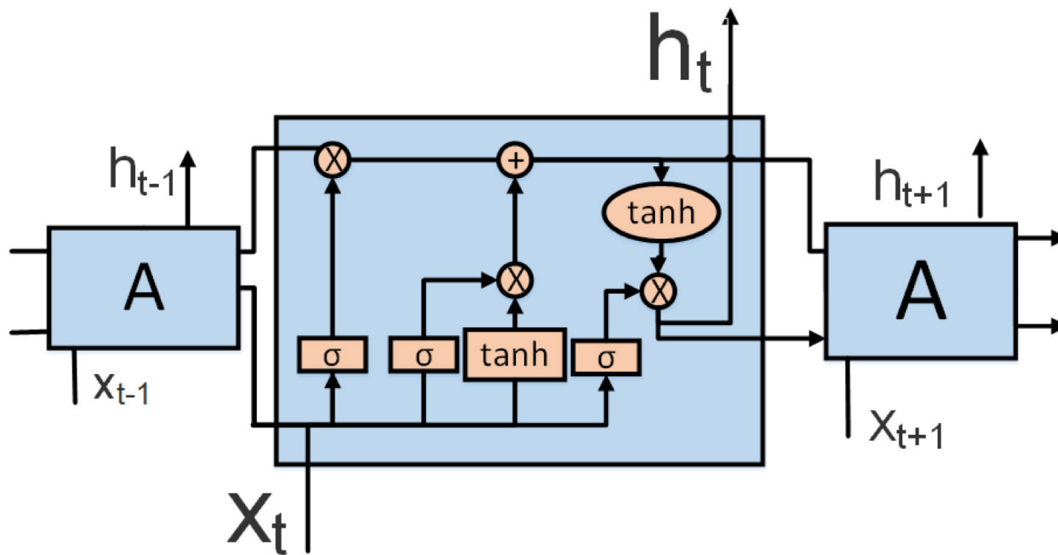


Figure 5: LSTM's cell state [45, 16]

features 3 gates: the forget gate, the input gate, and the output gate. The forget gate is used to discard irrelevant information. The input gate is used to add new information. The output gate determines the output.

3 Requirements

3.1 Overall Project Requirements

The overall goal of this project is to create a tool for processing and searching tweets, webpages, and ETDs for various information. Each content area is handled by a separate team of students. In addition, there are two teams involved in frontend design and integration. The frontend team is responsible for building the user-interface by which an end-user will search for information. The integration team is responsible for setting up virtual machines to make Ceph and NFS accessible to teams that need it, setting up containers for ElasticSearch, and connecting the Docker containers created by other teams to a local testing cluster. The tweet, webpages, and ETD teams will collaborate with the frontend team to determine what search functionality will be available and how data will be delivered to the end-user.

3.2 ETD Team Requirements

The goals of the ETD Team are as follows:

- Extract metadata from the repository of ETDs and ingest it into Elasticsearch.
- Extract full text from the repository of ETDs and ingest it into Elasticsearch.
- Extract chapters from the repository of ETDs and ingest it into Elasticsearch.
- Extract figures and tables from the ETDs and ingest them into Elasticsearch.
- Classify each ETD and its chapters to increase accessibility.
- Collaborate with the FE team to provide Elasticsearch endpoints for searching ingested data.
- Collaborate with the INT team to set up Docker containers for the major tasks in our workflow.

4 Design

4.1 Data Extraction

4.1.1 Tools for Extraction

There are many libraries for extracting data from PDF files. Different tools are suited to extracting different kinds of information. Moreover, each tool requires a different level of configuration and produces a different output format. In this section we highlight the extraction tools/libraries that we have utilized to parse information from the ETDs.

GROBID Generation Of Bibliographic Data (GROBID) is a machine learning library for parsing PDF files into XML/TEI encoded documents [26]. Metadata was extracted from these XML documents into text documents.

PyPDF2 PyPDF2 is a Python library with the ability to extract information from PDF files and manipulate them through splitting, merging, cropping, and transforming [38]. The fulltexts of ETDs were extracted using PyPDF2.

ITCore Intelligent Textbooks Core (ITCore) is a Java based tool that processes PDF files, particularly textbooks. Based on Jude's work, ITCore can be modified to better suit ETDs [17]. For each ETD, it produces a TEI knowledge model and segmented chapter PDF files and text files.

Extracting Tables and Figures: Neural Networks and YOLOv5 Our initial efforts in figure and table extraction were built on PyMuPDF and pdfplumber, respectively. As discussed in the literature review, both of these tools are intended to work with born-digital documents. While these tools were sufficient to build a fragile extraction service, we feared that the performance on scanned ETDs would be unacceptably poor. Fortunately, Kahu created a robust figure extraction tool as part of his recent thesis [19]. Instead of inferring the position of tables and figures from the syntax of the PDF file, Kahu's system utilizes a deep neural network based on the YOLOv5 object detection framework [35].

As mentioned in Section 1.4, the current pipeline does not feature a full adaptation of Kahu's work [18]. Kahu's pipeline, which is itself an extension of deepfigures, contained some execution patterns that were troublesome to port to the microservices production environment. We were able to isolate the core inference functionality and support it with some auxillary services of our own creation. Whereas Kahu's pipeline opts to use Ghostscript to render PDF files in an image format that is supported by the detection model, we provide a microservice based on pdf2image, a library that converts from PDF files into a PIL image object. The core inference service takes these images as input and outputs bounding boxes of any figures it detects within the input images. A third service uses Python [14] to crop the bounding boxes into their own PNG files as a final output.

4.1.2 Metadata Extraction

As mentioned in prior sections, teams from previous semesters of CS 5604 have already extracted metadata using GROBID. The extraction was imperfect and resulted in some mismatches between

the original PDF files and the resulting metadata. In this section we first describe the quality of the already extracted metadata, then discuss further steps for extraction. Table 2 is a listing of the current metadata fields and their descriptions.

To determine the quality of the prior metadata we manually inspected the metadata of about 410 master's theses[2]. Within this subset of the metadata, 76 records (18.53%) have some form of mismatch between the original PDF files and the corresponding metadata. The most common mismatches occur within the document's title and any fields that describe names. The mismatches in names mostly come from two cases. (1) A name in the PDF file shows their middle initial while the records show the full middle name. (2) A name in the PDF file shows a person's nickname while the corresponding record shows the full name. The mismatches in title are from records containing misspelled words or omitted words. Figures 6 and 7 show examples of these cases.

Host-Parasitoid Interactions of Two Invasive Drosophilid Species in Virginia Fruit Crops

title-none: Host-Parasitoid Interactions of Two Invasive Drosophilids in Virginia Fruit Crops

Figure 6: A mismatch where the metadata record is missing the word "species" in the corresponding field

U.S. Senate Deliberations on the War Powers during the Bush and Obama Administrations

title-none: U.S. Senate Deliberations on the War Powers Resolution during the Bush and Obama Administrations

Figure 7: A mismatch where the record has the more correct version of the title. For the purposes of searching, this is fine.

From this analysis, we determined that cleaning the extracted metadata was not our highest priority. However, since these mismatches can affect other systems, like our classifier, we marked the task of automatically verifying the metadata via a script as a future work.

4.1.3 Fulltext Extraction

Fulltext extraction was performed by the 2019 CME team. They utilized PyPDF2 to obtain text files containing the entire contents of ETDs [20]. We index these fulltexts into Elasticsearch and use them to classify the ETDs. However, PyPDF2 is no longer maintained, so we intend to use PyMuPDF for future text extraction.

[2] Instructions for accessing this data are listed in Section 8.

4.1.4 Chapter Extraction

ITCore [5] is one of the first information extraction tools that combines text, layout, document organization, and knowledge model based methods into one streamlined engine. The software uses a set of 39 rules in a linear workflow as shown in Figure 8. The four main stages of ITCore's workflow are as follows:

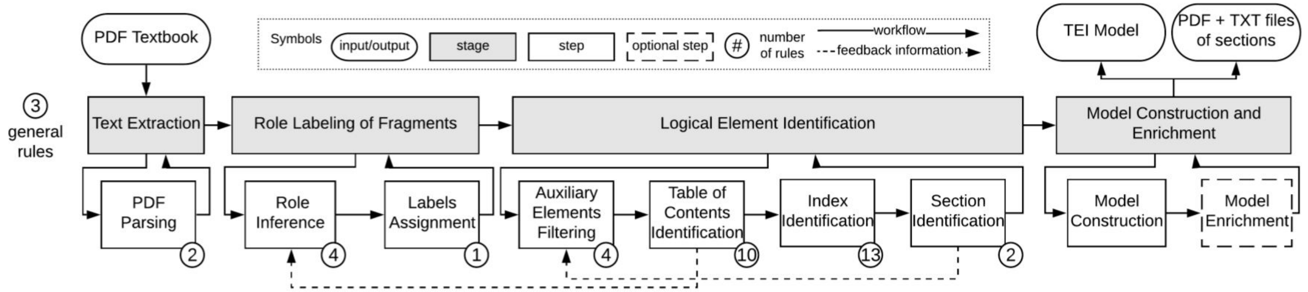


Figure 8: Stages and Steps in ITCore's Workflow

1. **Text Extraction**: Starting with PDF Parsing, each character is rule checked for rotation and order. During the *Rotated_ Coordinates* rule check, all groups of characters with a non-zero rotation are rotated to be horizontal. Then using bottom-up hierarchical grouping, a list of characters is grouped to form a list of words. The list of words is grouped to form a list of lines and finally the lists of lines is grouped to form a list of pages. These steps give us a list of characters $(c_1, c_2, ..., c_n)$ forming a word w, a list of words $(w_{x1}, w_{x2}, ..., w_{xn})$ forming a line l_x, a list of lines $(l_{y1}, l_{y2}, ..., l_{ym})$ forming a page p_y, and a list of pages $(p_1, p_2, ..., p_z)$ forming the document. This bottom-up grouping rule is called *Order_ Of_ Characters*.

2. **Role Labeling of Fragments**: Using clues from the layout and formatting, the role of each fragment is labelled. First, the order of all the fonts found in the document is determined with high font size, bold or italic font face, different font color, and low number of occurrences of the font ranked higher than their counterparts. This rule is called *Order_ of_ Styles*. Once ranked, the font associated with the body style is labelled based on the most occurring font style using the *Body_ Text_ Style* rule. Ignoring the body style font, the rest of the higher ranked fonts are labelled as a heading or subheading under the *Label_ Of_ Styles* rule check. To accommodate special texts, including quotes, formulae, or remarks, a further check, *Label_ Of_ Styles_ Cross_ Check*, is performed by deleting all fonts that are not found in the table of contents. Lastly, adjacent lines of text are grouped by font style to form fragments under the *Text_ Fragments* rule.

3. **Logical Element Identification**: During this stage all of the finer elements are identified through 4 steps:

   - **Auxiliary Elements Filtering**: Auxiliary Elements such as headers, footers, copyright text, and page numbers are eliminated by using a set of 4 rules: *Repeated_ Lines*,

Copyright_ Text, *Position_ Of_ Page_ Numbers* and *Missing_ Page_ Numbers*. This results in $M =< P, N >$, where M is storing the mapping, P represents the pages in the document, and N represents the order of the pages.

- **Table of Contents (TOC) Identification**: The beginning of the TOC is identified (*Beginning_ Of_ TOC*) and every page is grouped together (*TOC_ Page_ Layout*). Next, in order to identify the hierarchy of the TOC titles, the page margins are corrected so they line up accurately (*Page_ Margin_ Correction*) and multi-line entries are condensed to one line (*Multiline_ Entry*). This results in a list of each TOC entry, e, represented as $(e_0, e_1, ..., e_n)$.
  This list still has introductory section entries, author entries, and part entries (subsections of lower level). Each of these is identified by using regular expressions (*Introductory_ Entry*), a named-entity recognition (NER) algorithm (*Author_ Entry*), and the line spacing before and after an entry (*Part_ Entry*).
  There are 3 types of TOCs – flat, flat-ordered, and indented – which are identified by the *TOC_ Type* rule and passed to the corresponding *Hierarchy_ Of_ Entry* rule, resulting in a sorted list of entries. Given the order of the entries, the *End_ Of_ Section* rule returns the end of section page number as the next entry's start page number, which is found in the TOC by the *Hierarchy_ Of_ Entry* rule. This results in $M =< P, N, TOC >$, where TOC is the list of start and end page numbers of the sections based on the hierarchy of entries in the TOC.

- **Index Identification**: An index section is created to map relevant terms in the domain of the document to the page number where the term is introduced. The goal of this stage is to try to identify the index section within the document.
  Starting with the *Beginning_ Of_ Index* rule, it detects the beginning of the index section as defined in the TOC entries. After detecting the first index page, the rest of the index pages are determined using the rule, *Index_ Page_ Layout*, which looks for lines ending with page references and starting without a heading text fragment. Since index pages are often broken into multi-column layouts, we use the *Multicolumn_ Layout* rule to identify the different columns and correct for indentation within columns using *Column_ Margin_ Correction*.
  After identifying the index pages and columns, we begin to identify each index term. Using regular expressions, we identify single alphabet lines and remove them (*Alphabet_ Letter*). Each index has a heading and/or subheading, reference pages (addressed here as locators), and/or cross-references with "see" or "see also" words. These are identified by the following rules: *Locator_Element* uses regular expressions to identify the 5 types of page references, *Term_ Delimiter* uses word occurrence frequency to identify delimiters and thus headings and sub-headings, and *Cross_ Reference* looks for predefined cross-reference identifiers. Similar to the TOC, index terms can be multiline, which are identified using the *Multiline_ Term* check.
  Lastly, nested hierarchies are identified using the *Flush_ And_ Hang*, *Continued_ Terms*, and *Run-in_ Style* or *Indented_ Style* rules. *Flush_ And_ Hang* groups index terms, *Continued_ Terms* identifies any terms that start on one page and continue on to the next page using keywords such as "cont." or "continued", *Run-in_ Style* identifies groups of terms separated by a delimiter, and *Indented_ Style* identifies groups of terms separated

by indents. Lastly, the order of the words within an index term is determined using *Reading_ Label*. This results in $M =< P, N, TOC, I >$ where I is the list of index entries.

- **Section Identification**: A chapter can start mid-way through a page. In order to break the PDF file into chapters containing only the text within a given chapter (and not text on the same page that belongs to a different chapter) we need to identify the exact boundaries on the starting and ending page (*Section_ Content*); the starting and ending page are obtained from P and TOC in M. To connect the index terms associated with each section, we use I to find the index terms within the page bounds of each section (*Corresponding_ Section*). From these section mappings we get two lists, one mapping section segments into PDF files and another mapping section segments into texts, making $M =< P, N, TOC, I, PDF, TXT >$

4. **Model Construction and Enrichment**: The Text Encoding Initiative (TEI) format is used to create the model components: Structure, Content, and Domain Knowledge, from M. Once we have the TEI model, it can be linked to Open Data for additional semantic information [5]. For now, we have not explored the Model Enrichment abilities of ITCore.

4.2 Classification

This section describes our classification pipeline. We first applied the methods created by Jude [17] for classification, then developed an alternative method for classifying the ETDs in an effort to increase quality as indicated by metric ratings.

4.2.1 Categorical System

This section describes the category system our classifiers will be using. The main goal was to make each ETD within the VTechWorks repository as accessible as possible. Since a single ETD can contain information across many different disciplines, it is crucial to classify each ETD at a chapter-level granularity. The challenge, however, is that there is no universally followed standard of categories within ETD classification. Since we want our system to be publicly accessible, having a standard category system is mandatory. The categories within the system had to be something that could represent most disciplines within the vast ETD landscape. For this task, Jude leveraged the ProQuest Subject Category system to build a standard that could easily represent a majority of all disciplines [17].

Categories
Adult education
Aerospace engineering
Biomedical engineering
Chemical engineering
Civil engineering
Computer Engineering
Computer science
Ecology
Educational leadership

Educational psychology
Electrical engineering
Elementary education
Environmental science
Forestry
Higher education
Industrial engineering
Marketing
Materials science
Mathematics
Mechanical engineering
Molecular biology
Occupational psychology
Organic chemistry
Public administration
Secondary education
Special education
Statistics
Teacher education

Table 1: Categories (based on the ProQuest Subject Category system)

Table 1 shows the categories Jude consolidated as the most representative. We also use these categories for classification. Jude chose these categories to be the representative set after performing an extensive exploratory data analysis of the distribution of ETDs.

Figure 9 shows the top 25 categories of ETDs from Virginia Tech, Penn State, and University of Illinois at Urbana-Champaign.

ProQuest Open ProQuest Dissertations and Theses (PQDT) Open provides free access to full ETDs with the aim of growing the reach of each publisher's work. PQDT Open has a vast collection of ETDs from many different schools and disciplines. It also include many ETDs from Virginia Tech students as well. The PQDT system also allows users submitting an ETD to provide additional information in the form of additional categories and keywords to help further categorize the ETDs within their system. PQDT Open represents a diverse set of ETDs that provides us with a large and diverse dataset to help train our classifiers.

4.2.2 Classification using Jude's Method

This section describes the first method we implemented for our classification system. The pipeline in this method follows the work done by Jude.

Figure 10 shows the full pipeline Jude used to classify ETDs on fulltext and chapter-text. We implemented this same pipeline in order to classify ETDs using fulltext. The following sections

Virginia Tech ETD(s)		Pennsylvania State University ETD(s)		University of Illinois at UrbanaChampaign ETD	
Electrical and Computer Engineering	711	Mechanical Engineering	481	Electrical & Computer Eng	848
Educational Leadership and Policy Studies	611	Electrical Engineering	449	Computer Science	604
Chemistry	577	Computer Science and Engineering	341	Mechanical Sci & Engineering	460
Mechanical Engineering	499	Industrial Engineering	321	Civil & Environmental Eng	385
Civil Engineering	403	Chemistry	317	Chemistry	278
Psychology 398	398	Aerospace Engineering	297	Physics	240
Educational Administration	369	Materials Science and Engineering	294	Psychology	223
Teaching and Learning	363	Psychology	244	Aerospace Engineering	214
Industrial and Systems Engineering	326	Geosciences	194	Educ Policy, Orgzn & Leadrshp	196
Computer Science	299	Energy and Mineral Engineering	187	Animal Sciences	189
Engineering Mechanics	271	Civil Engineering	185	Crop Sciences	177
Electrical Engineering	249	Curriculum and Instruction	177	Mathematics	169
Mathematics	246	Information Sciences and Technology	174	Materials Science & Engineerng	149
Curriculum and Instruction	238	Physics	164	Kinesiology & Community Health	143
Aerospace and Ocean Engineering	234	Engineering Science and Mechanics	156	Natural Res & Env Sci	121
Statistics	233	Human Development and Family Studies	152	Nuclear, Plasma, & Rad Engr	115
Chemical Engineering	224	Chemical Engineering	148	Agr & Consumer Economics	102
Engineering Science and Mechanics	215	Nuclear Engineering	136	Chemical & Biomolecular Engr	98
Physics	206	Statistics	136	Educational Psychology	96
Human Development	186	Mathematics	133	Music	94
Vocational and Technical Education	175	Economics	133	Curriculum and Instruction	91
Economics	173	Biochemistry, Microbiology, and Molecular Biology	132	Bioengineering	90
Biology	172	Architectural Engineering	129	Engineering Administration	89
Public Administration and Public Affairs	168	Kinesiology	126	Food Science & Human Nutrition	89
Entomology	148	Adult Education	119	School of Molecular & Cell Bio	88

Figure 9: ETD Collections from VT, Penn State, and UIUC

describe the parts of the pipeline in more detail. The PQDT dataset shown in the figure above was provided by our SME, who worked closely with Jude.

The first step in the pipeline was to preprocess the data and get it ready for our machine learning classifiers. For this, Jude decided to embed the longer-text differently than the metadata fields.

Long-Text Embedding Traditional embedding models like Word2Vec and fastText are not ideal for longer-text. A single phrase can be used in different contexts depending on the section it resides in. Doc2Vec was designed to help embed longer-text. As discussed in Section 2, Doc2Vec builds upon the Word2Vec model. Word2Vec's continuous bag of words technique creates a sliding window of words around a target word to help build context around the target word. Doc2Vec builds upon the sliding window by adding a paragraph ID to it as well. This helps maintain context throughout a long document. Since Doc2Vec was well suited for longer-text, Jude used it to embed the abstract and fulltext of each document. To help fine tune the classifiers, Jude experimented with varying the set of Doc2Vec hyper-parameters. The main parameters that were tweaked are as follows:

- **dm:** Flag to use Distributed Bag-Of-Words or Distributed Memory
- **window:** How many words around a target to use for context
- **dim:** Length of the feature vector to generate for each document
- **epochs:** Number of iterations to train the deep learning model

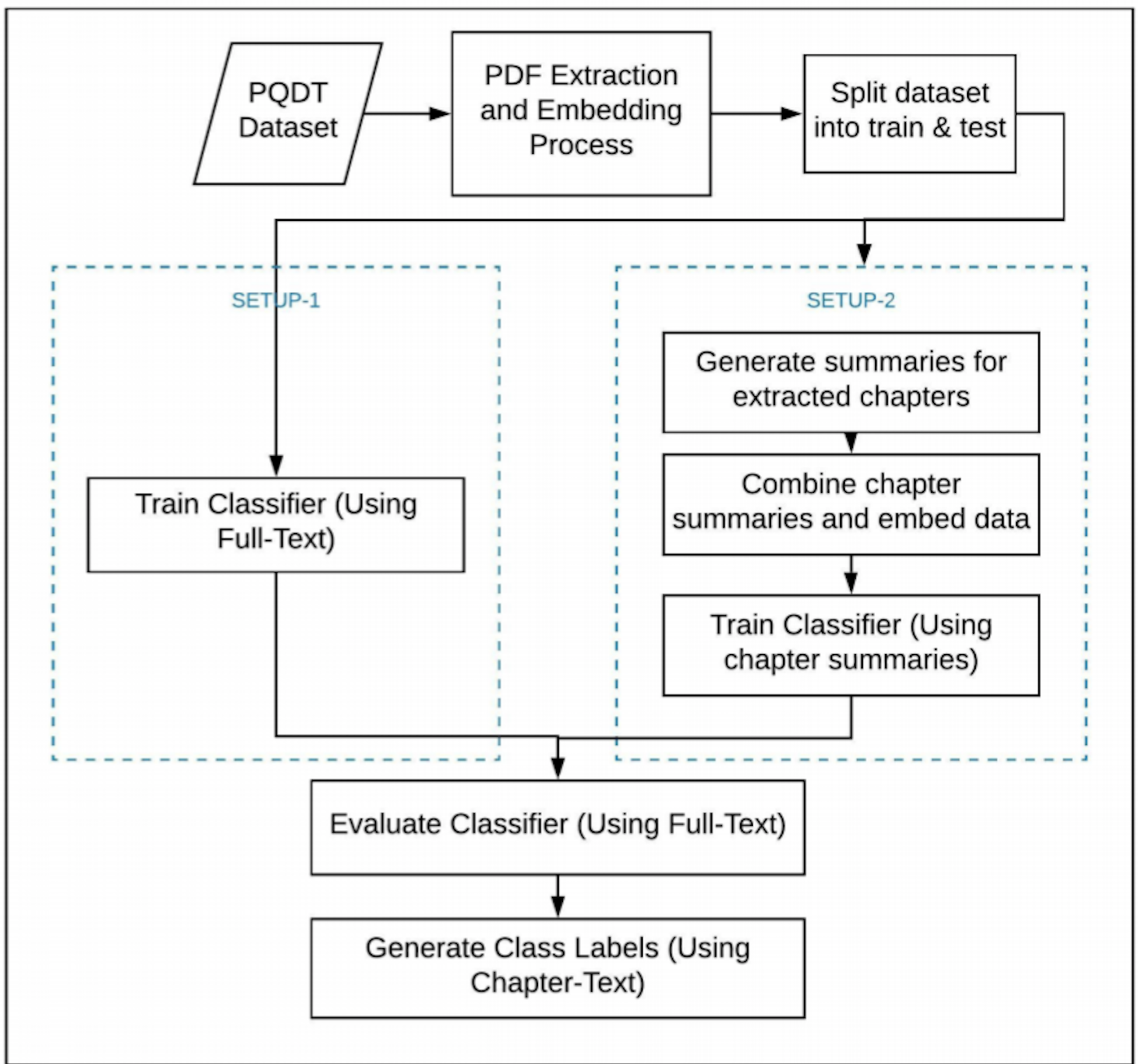


Figure 10: System Overview: Classification Pipeline [17]

The following shows the list of Doc2Vec models created by varying the hyper-parameters:

The machine learning models were trained using each of the embedding models listed in Figure 11 and the models generated metrics that helped fine-tune our embedding models.

Metadata Embedding We only needed to use Doc2Vec for our abstract and fulltext. All of the other metadata fields used in our classifier could be easily embedded using any of the traditional embedding models. For the task, Jude chose to embed her metadata using Facebook's fastText. With fastText, however, we only used one set of hyper-parameters, which are detailed in Figure

```
'0dm_8win_100dim_100e',
'0dm_8win_100dim_25e',
'0dm_8win_100dim_50e',
'0dm_8win_200dim_100e',
'0dm_8win_200dim_25e',
'0dm_8win_200dim_50e',
'1dm_8win_100dim_100e',
'1dm_8win_100dim_25e',
'1dm_8win_100dim_50e',
'1dm_8win_200dim_100e',
'1dm_8win_200dim_1e',
'1dm_8win_200dim_25e',
'1dm_8win_200dim_50e',
'0dm_8win_100dim_100e',
'0dm_8win_100dim_25e',
'0dm_8win_100dim_50e',
'0dm_8win_200dim_100e',
'0dm_8win_200dim_25e',
'0dm_8win_200dim_50e',
'1dm_8win_100dim_100e',
'1dm_8win_100dim_25e',
'1dm_8win_100dim_50e',
'1dm_8win_200dim_100e',
'1dm_8win_200dim_1e',
'1dm_8win_200dim_25e',
'1dm_8win_200dim_50e']
```

Figure 11: Doc2Vec embeddings generated by tuning hyper-parameters

12.

```
{

  Epochs: 100,
  WordNGrams : 3,
  Dimensions : 100

}
```

Figure 12: fastText Embedding Model parameters

Machine Learning Models As discussed in the literature review, our assumption is that a single ETD can relate to multiple different disciplines. This makes our classification problem a multi-label classification problem. Multi-label classifiers often are built upon the more traditional

machine learning models such as Random Forest, Support Vector Machines (SVM), and Logistic Regression.

Among the many different multi-label classifiers discussed in the literature review, Jude decided to create her multi-label classifier upon the **Label Powerset** classifier. Label Powerset creates a single "power-set" of all categories and then trains this classifier on all unique category combinations found within a dataset. Since the classifier trains on all category combinations, it also takes into account possible correlations between categories. One thing to note is that while using a Label Powerset, a baseline "traditional" machine learning model needs to be chosen to use for eachcategory combination it finds. To that end, Jude chose to test the Label Powerset using SVMs and RF.

```
['RF_D#0dm_8win_100dim_100e#F#100emodel3N100D',
 'RF_D#0dm_8win_100dim_25e#F#100emodel3N100D',
 'RF_D#0dm_8win_100dim_50e#F#100emodel3N100D',
 'RF_D#0dm_8win_200dim_100e#F#100emodel3N100D',
 'RF_D#0dm_8win_200dim_25e#F#100emodel3N100D',
 'RF_D#0dm_8win_200dim_50e#F#100emodel3N100D',
 'RF_D#1dm_8win_100dim_100e#F#100emodel3N100D',
 'RF_D#1dm_8win_100dim_25e#F#100emodel3N100D',
 'RF_D#1dm_8win_100dim_50e#F#100emodel3N100D',
 'RF_D#1dm_8win_200dim_100e#F#100emodel3N100D',
 'RF_D#1dm_8win_200dim_1e#F#100emodel3N100D',
 'RF_D#1dm_8win_200dim_25e#F#100emodel3N100D',
 'RF_D#1dm_8win_200dim_50e#F#100emodel3N100D',
 'SVM_D#0dm_8win_100dim_100e#F#100emodel3N100D',
 'SVM_D#0dm_8win_100dim_25e#F#100emodel3N100D',
 'SVM_D#0dm_8win_100dim_50e#F#100emodel3N100D',
 'SVM_D#0dm_8win_200dim_100e#F#100emodel3N100D',
 'SVM_D#0dm_8win_200dim_25e#F#100emodel3N100D',
 'SVM_D#0dm_8win_200dim_50e#F#100emodel3N100D',
 'SVM_D#1dm_8win_100dim_100e#F#100emodel3N100D',
 'SVM_D#1dm_8win_100dim_25e#F#100emodel3N100D',
 'SVM_D#1dm_8win_100dim_50e#F#100emodel3N100D',
 'SVM_D#1dm_8win_200dim_100e#F#100emodel3N100D',
 'SVM_D#1dm_8win_200dim_1e#F#100emodel3N100D',
 'SVM_D#1dm_8win_200dim_25e#F#100emodel3N100D',
 'SVM_D#1dm_8win_200dim_50e#F#100emodel3N100D']
```

Figure 13: Label Powerset Models Generated

Figure 13 is the list of all the multi-label classifiers we generated using the set of Doc2Vec and

fastText embeddings we generated earlier. For each model, we generated metrics to help evaluate which combination of embedding hyper-parameters and baseline model performed best.

4.2.3 Classification Using Probabilistic-Based ETD Classification

The low metrics from the models generated in the first approach (discussed in Section 6) led us to research applying a new approach to classification. Our primary motivation was to train our classifiers using focused text instead of using the fulltext to help better represent each class. Our assumption has been that an ETD can discuss many different domains at the same time. Hence, we decided to design a new pipeline and test its metrics against the one previously discussed.

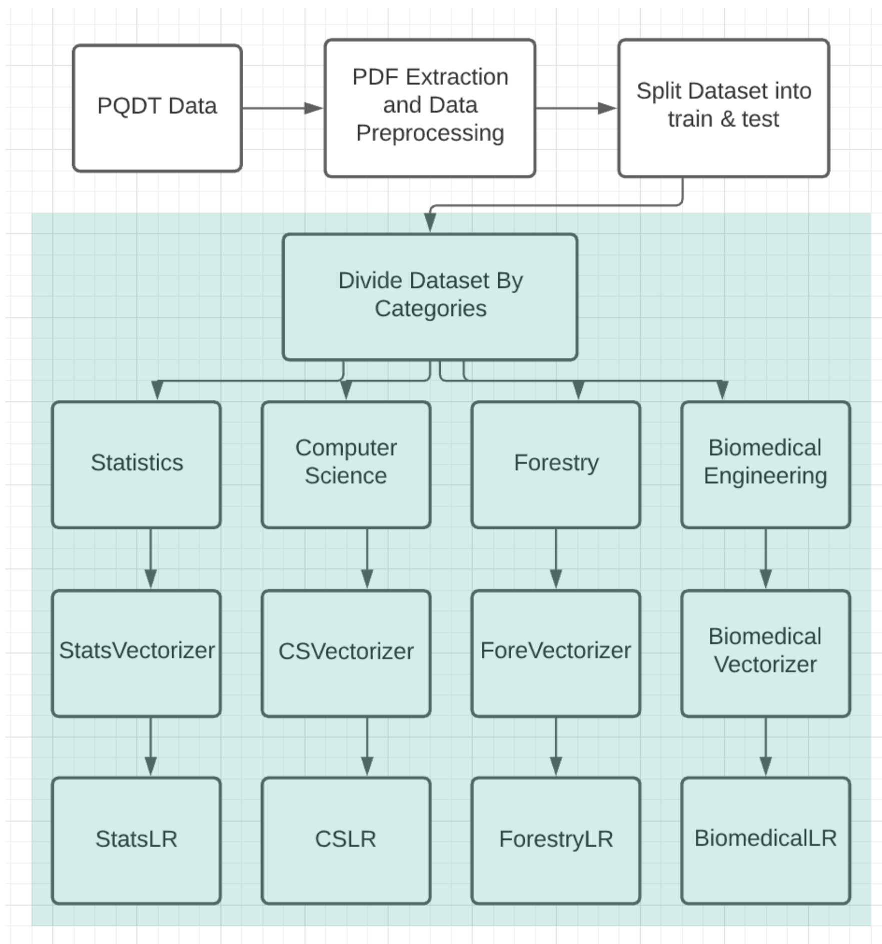


Figure 14: System Overview: New Classification Pipeline

Figure 14 shows the design of training our classifiers in the new pipeline. We chose each ETD's title, keywords, and abstracts as our focused text to train on. Our PQDT dataset is divided by each category, and for each category's dataset, we train an individual TF-IDF vectorizer and logistic regression model.

Dataset TF-IDF We divided the data-set into smaller data-sets that represent each category. For each category we generated term frequency - inverse document frequency values for the cor-

pus. The intuition behind this was that the collection of focused text will represent the term's frequency, weights, and out-of-vocabulary (OOV) words better then using the fulltext. We trained our vectorizer with the parameters in Figure 15.

```
analyzer = "word"
norm =  "l2"
ngram_range = (1,3)
```

Figure 15: TF-IDF Vectorizer params

The **analyzer** parameter determines the scope of the vectorizer. When set to *word*, it uses a word as a token instead of a character. We chose the granularity of our token to be *word* because we want to combine it with the n-gram range to catch phrases instead of single word frequencies. With our current n-gram range (1,3), a sentence such as "This is a cute dog" will result in the n-grams "This is a", "is a cute", and "a cute dog". We wanted the vectorizers to be able to separate word context from among different domains. A good example would be the use of the word "neural" in medical applications versus its use in computer science (in neural networks).

Machine Learning Models Instead of using a multi-label classifier for our project, we used individual logistic regression models for each category. We chose logistic regression models as our baseline models because we wanted to retrieve the probability estimate for each category. We then trained each category's machine learning model, applying its own vectorizer on the entire corpus's focused text.

```
C = 1.25
solver =  "liblinear"
```

Figure 16: Logistic Regression Model Params

Figure 16 shows the parameters we trained our logistic regression models on. One thing we noticed while varying some of the hyper-parameters was that increasing C would increase the ability of each department's model to discern itself better.

Example code for generating prediction probabilities for a statistics abstract is shown in Figure 17. Here, b is the text of the statistic's focused text vectorized using the vectorizer for each category. The second value in each outputted array is the probability ofthe document belonging to that category. As you can see, the statistics model has the highest probability compared to the other categories, with electrical engineering being up next.

Generating Full ETD Labels The next step was applying the classifiers on the fulltext. Since we trained our classifiers using focused text that has a much smaller text body compared to the fulltext of an ETD, we wanted to apply our classifiers over similar smaller segments of the ETD and cumulatively generate the most probable label for an ETD.

```
## Prediction Probablities For b in Statistics
modelByCat["Statistics"]["model"].predict_log_proba(b_S)
```

```
array([[-0.15013792, -1.97033083]])
```

```
## Prediction Probablities For b in Forestry
modelByCat["Forestry"]["model"].predict_log_proba(b_F)
```

```
array([[-0.00992365, -4.61779255]])
```

```
## Prediction Probablities For b in Adult Education
modelByCat["Adult education"]["model"].predict_log_proba(b_A)
```

```
array([[-0.01929883, -3.9573445 ]])
```

```
## Prediction Probablities For b in Electrical Engineering
modelByCat["Electrical engineering"]["model"].predict_log_proba(b_E)
```

```
array([[-0.08698939, -2.48514856]])
```

Figure 17: Logistic Regression Model Predictions On A Statistics ETD

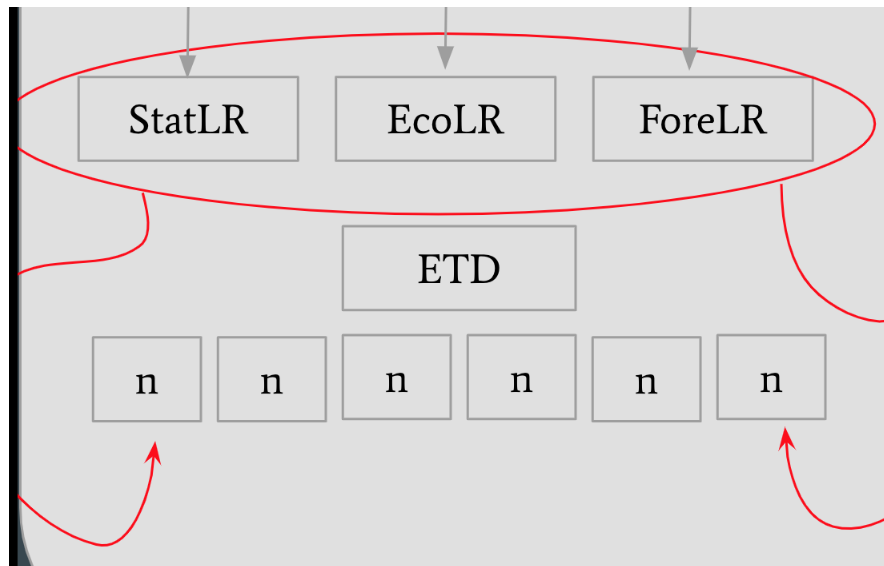


Figure 18: Dividing and training ETD

Figure 18 displays the method of dividing up a single ETD into n segments and applying the category models on each segment. We generated cumulative probabilities for the chance of that category being true for all categories across each segment and chose the highest probable categories as representative of an individual ETD.

4.3 ETD Metadata Fields

Table 2 describes the fields in our metadata records. The metadata is based on a combination of Dublin Core metadata terms along with additional useful fields relating to ETDs. These fields also formed the basis of our Elasticsearch mapping for metadata. A listing of the equivalent metadata mapping passed to Elasticsearch can be seen in Appendix A.1.

Field	Description
contributor_author	main author of the document
contributor_committeechair	chair(s) of the author's advisory committee
contributor_committeecochair	co-chair(s) of the author's advisory committee
contributor_committeemember	member(s) of the author's advisory committee
contributor_department	the major/department that the author belongs to
date_accessioned	date the document was added to the collection
data_available	date the document became available for viewing
date_issued	date of formal issuance
date_adate	unknown
date_sdate	unknown
date_rdate	unknown
degree_discipline	degree awarded
degree_grantor	entity that granted the degree
degree_level	level of degree; usually "doctoral" or "masters"
degree_name	name of degree; usually "PHD" or "MS"
description_abstract	a document's abstract
description_degree	degree; either "MS" or "PHD"
description_provenance	any changes to a documents ownership that are noteworthy
description_sponsorship	any notes on entities who have sponsored the research
format_medium	type of document; always "ETD"
handle	Unique identifier for documents
identifier_other	unique identifiers for the document that are not the handle
identifier_uri	a link to the actual document
publisher	entity responsible for making a resource available
relation	handle(s) of related document(s)
relation_haspart	files that make up the original document
rights	any copyright or other ownership information
subject	a list of subject/keywords describing the document
subject_lcc	Library of Congress categories
subject_lcsh	Library of Congress subject headers
title	the title of the document

Table 2: ETD Metadata Fields

4.4 ETD Processing Pipeline

In order to process ETD data, we implemented a pipeline of Docker containers that would each represent a microservice. A diagram of the entire pipeline is shown in Figure 19. Input and output to each microservice is controlled via environment variables. In the production environment, the user controls the pipeline through an interface created by the FE team. The parameters they pass in, as well as extra variables defined by our pipeline, are set by Airflow when spawning a container.

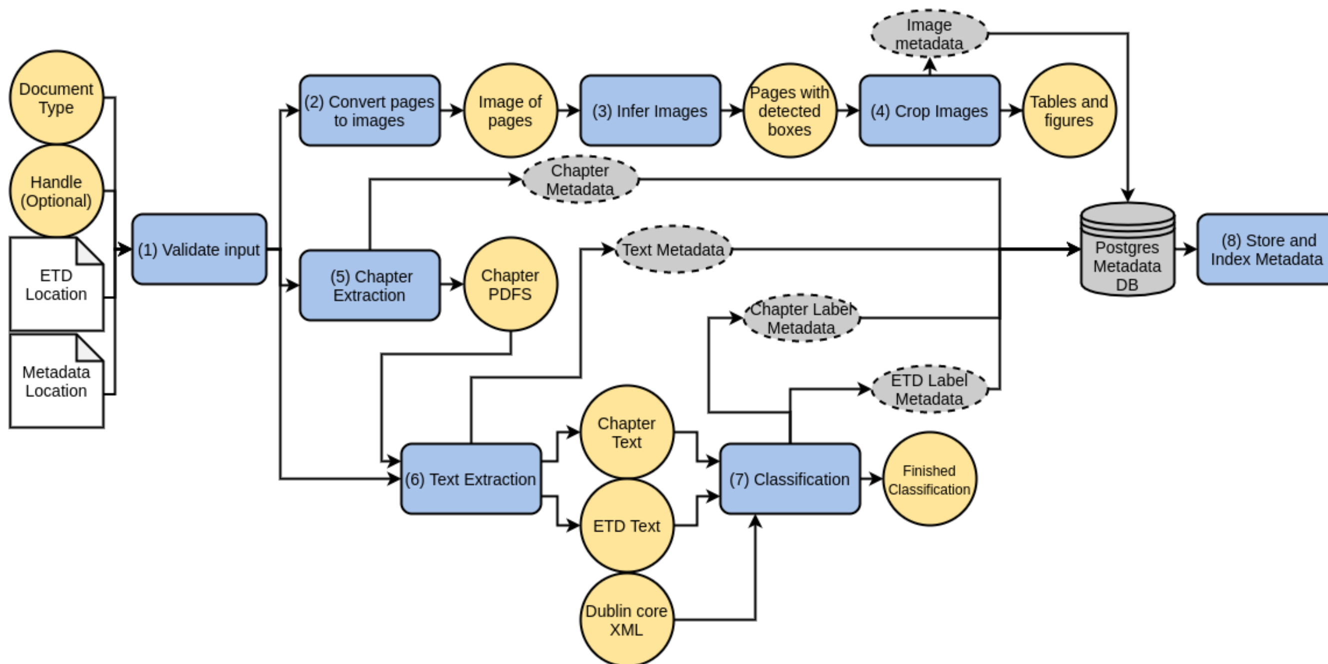


Figure 19: Pipeline for processing ETDs

In the following sections, we describe the main sections in our data processing pipeline. Each subsection contains a high-level description of the service(s), the types of data that are used and created by each service, and the operations performed by each service.

4.4.1 Validate Input

The validation microservice is responsible for taking the input metadata, document type, and handle as input and storing that in a Postgres database. This serves the purpose of creating an entry in the database that the other microservices can update with additional information. Although this service takes in the original ETD PDF file as an input parameter, the file itself is not used in any way by the microservice. The PDF file is specified as a input parameter here because of the overall Airflow pipeline. This ensures that the validate service runs before any other microservice in the pipeline and prevents other microservices from trying to update database records that do not yet exist.

4.4.2 Image Extraction

The process of image extraction is separated into 3 microservices:

PDF to Image The first microservice in this section converts each page of a PDF file into a PNG image. This conversion needs to take place because the subsequent image detection service does not take PDF files as input. This microservice accepts an ETD as input and turns each page into a PNG file. The output pages are stored in a folder specified by the input parameters.

Image Inference The second microservice takes a directory of PNG files and searches for figures and tables. Currently the service will produce two files for each page: a PNG file with boxes drawn around the detected images and a text file denoting the coordinates of each box on said page. Note that storing the pictures of pages with boxes around the images is optional. Since the coordinates of each box are saved in a corresponding text file, it is feasible to reuse the input PDF files for the image cropping microservice. However, we decided that since there are a few cases where the image detection messes up, it would be better to allow the end user to visually see the results of detection.

Image Cropping The final microservice devoted to image extraction takes the pages with boxes and uses the corresponding text file of coordinates to crop the images. This process produces a folder with each of the detected images cropped out of the pages. The directory where the images are stored is added to the document's metadata in Postgres.

4.4.3 Chapter Segmentation

The chapter segmentation microservice takes an ETD and separates it into PDF files for each chapter using the techniques described in Section 4.1.4. The resulting chapter PDF files are stored in a directory and the location of the directory is added to the document's metadata in Postgres.

4.4.4 Text Extraction

This microservice extracts fulltext at the ETD and chapter-level. Input to this service can either be full ETDs or pre-segmented information. This process takes the input PDF files and extracts all of the text into .txt files. Note that this microservice has to be run twice due to the limitations of Airflow. Airflow was limited to handle only one output goal at a time for each service. For more information contact the INT team. The first invocation processes the full ETD and the second invocation processes the text of each chapter.

4.4.5 Classification

The classification service accepts multiple forms of ETD data and classifies using the techniques discussed in Section 4.2.3. The results of the classification are stored in the Postgres database and a simple JSON file with the contents `'{"classify_finish":"True"}'` for Airflow.

4.4.6 Store and index metadata

The final microservice takes a handle ID and document type as input. Using those two pieces, the service will retrieve the corresponding metadata record from the database and convert the record into a JSON structure that is compatible with the Elasticsearch index mapping shown in Appendix A.1. The only information that needs special processing are the chapter metadata and

the metadata fields that allow multiple values (e.g., subject). The chapter metadata is turned into a structured object to match the Elasticsearch specification. The fields that allow multiple values are split on each semicolon delimiter and turned into an array of values. Note that the use of the semicolons as a delimiter is arbitrary and initial investigation showed that the fields being split do not contain semicolons within themselves. This converted JSON is then sent to Elasticsearch.

5 Timeline

Table 3 shows the milestones for each of our services, as well as overall milestones. The table is broken up by service. It contains the task description, completion date / expected completion date (DNF means that the goals were not fully met), and the team member(s) responsible for accomplishing the task. This schedule has been added to and changed over the course of the project.

Table 3: Tasks and Timeline

Date	Task	Assignee
Overall Milestones		
09/14	Send Project Workflow artifacts to FE and INT teams	ALL
09/17	Interim Report 1	ALL
10/08	Interim Report 2	ALL
10/29	Interim Report 3	ALL
11/07	Talk with FE team about exposed services for workflow page	ALL
11/15	Test a service (any will do) using a docker container with mounted volumes	ALL
12/02	Final Presentation	ALL
12/07	Complete pipeline tested and deployed	ALL
12/09	Final Project Report	ALL
Literature Review		
09/20	Research previous teams' works and related topics	ALL
09/09	Talk to SME about goals, requirements, and prior works	ALL
Figure/Table Extraction		
10/14	PyMuPDF-based Figure Extraction microservice	Furman
10/14	PDFPlumber-based Table Extraction microservice	Furman
11/25	Robust Figure and Table Extraction Microservice (Leverage Kahu's Work)	Furman
11/29	Service Dockerized and Tested	Furman
DNF	Service for Training Figure/Table Extraction System (stretch goal)	Furman
DNF	Service for Evaluating Figure/Table Extraction System (stretch goal)	Furman
Chapter Segmentation		
10/12	Perform chapter extraction on data sets with ITCore	Nguyen, Raghuraman
10/29	Initial chapter extraction API	Nguyen, Raghuraman
12/05	Service dockerized and tested	Nguyen, Raghuraman

Continued on next page

Table 3 – continued from previous page

Date	Task	Assignee
Classification		
10/15	Get Palakh Jude's code working	Manzoor
10/28	Initial Classification API	Manzoor
11/20	Service dockerized and tested	Manzoor
Elasticsearch		
10/07	Initial Elasticsearch collection mapping created	Hardy
10/11	Elasticsearch microservice can be used to index into elasticsearch	Hardy
10/12	Elasticsearch microservice tested by indexing a subset of documents	Hardy
11/01	Elasticsearch microservice can be used to modify records in place	Hardy
11/03	Elasticsearch container dockerized and tested	Hardy
11/15	Elasticsearch microservice completed	Hardy
Metadata Extraction		
09/30	Assess quality of current metadata and compile examples of outliers	Fan
10/05	Determine whether the current metadata can be used for indexing while verification happens	Fan
DNF	Build tools to verify current metadata (stretch goal)	Fan
DNF	Index/reindex any metadata that was updated during the verification process (stretch goal)	Fan
Fulltext Extraction		
11/10	Initial Fulltext Extraction microservice	Fan, Raghuraman
12/05	Service dockerized and tested	Fan, Raghuraman

6 System Evaluation

6.1 Chapter Extraction

To evaluate the quality of chapter extraction, we used a subset of records from 2017.[3] We performed chapter extraction on the subset of 281 dissertations which resulted in 174 processed dissertations. This discrepancy is due to ITCore's inability to process ETDs with irregularly formatted tables of contents. Figure 20 is an example of a dissertation segmented into 6 chapter PDF files.

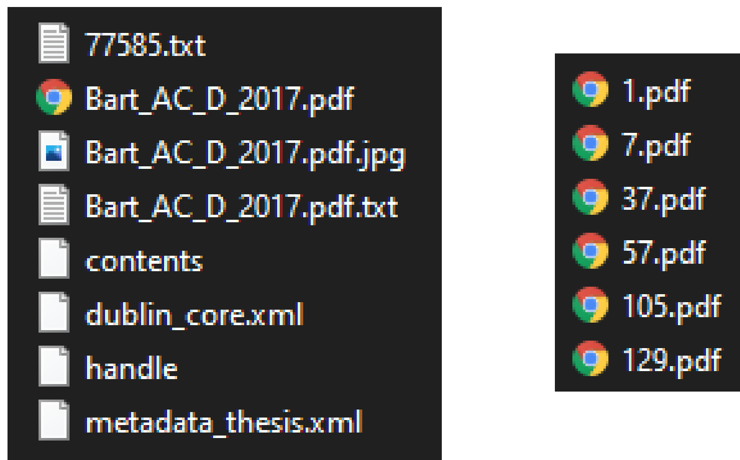


Figure 20: Example Segmented ETD. The left image is a snapshot of the the example dissertation's files. `Bart_AC_D_2017.pdf` is the original dissertation and `dublin_core.xml` contains its metadata. The right image depicts the resultant chapter PDF files. The chapter PDF files are named based on indices internal to ITCore and have no correlation to the page numbers of the original PDF.

In order to validate the results from chapter extraction, we need to have a set of actual knowledge models for the PDF files depicting the true number of chapters and their respective chapter boundaries for each PDF file. The actual knowledge models can only be calculated manually. The two main criteria we consider are:

- Number of chapters extracted: Were the segmented number of chapters within 10% of actual number of chapters in the ETD? For example, if we have 20 chapters in an ETD and segmentation through ITCore yields 18 chapters, then we are within 10% of the actual number of chapters.
- The start and end index of each chapter: Count of matches and mismatches for the page number where the chapter begins and ends. As some chapters could be skipped completely, we will look at both one-to-one matches and matches based only on the unordered start and

[3] The subset was curated by a previous CS5604 team.

end indices. To illustrate, consider an ETD with 3 chapters with the following start and end indices (3, 5), (5, 21), and (22, 30). ITCore may create PDF segments with the following start and end indices (3, 5), (5, 9), (10, 21), (22, 30). All of the start and end indices are captured as segment boundaries, but, if we consider one to one mapping of the segments, we only have two matches. We will thus have two scores for the segment boundaries:

1. Count of Segment Indices matched
2. Count of (start, end) index pairs matched

The above criteria can be applied to a subset of the ETDs. This gives us an idea of the distribution of the number of chapters that our ETD population should follow. For our complete data set, we will evaluate how well the distribution of the number of chapters aligns to the distribution of the actual/predicted number of chapters from the manually evaluated subset. We also planned to use common text segmentation evaluation metrics such as WindowDiff [33] and Boundary Edit Distance based boundary Similarity (BED-S) [13], but did not have the time to work on these metrics.

On the subset of dissertations, ITCore produces the following distribution of chapters:

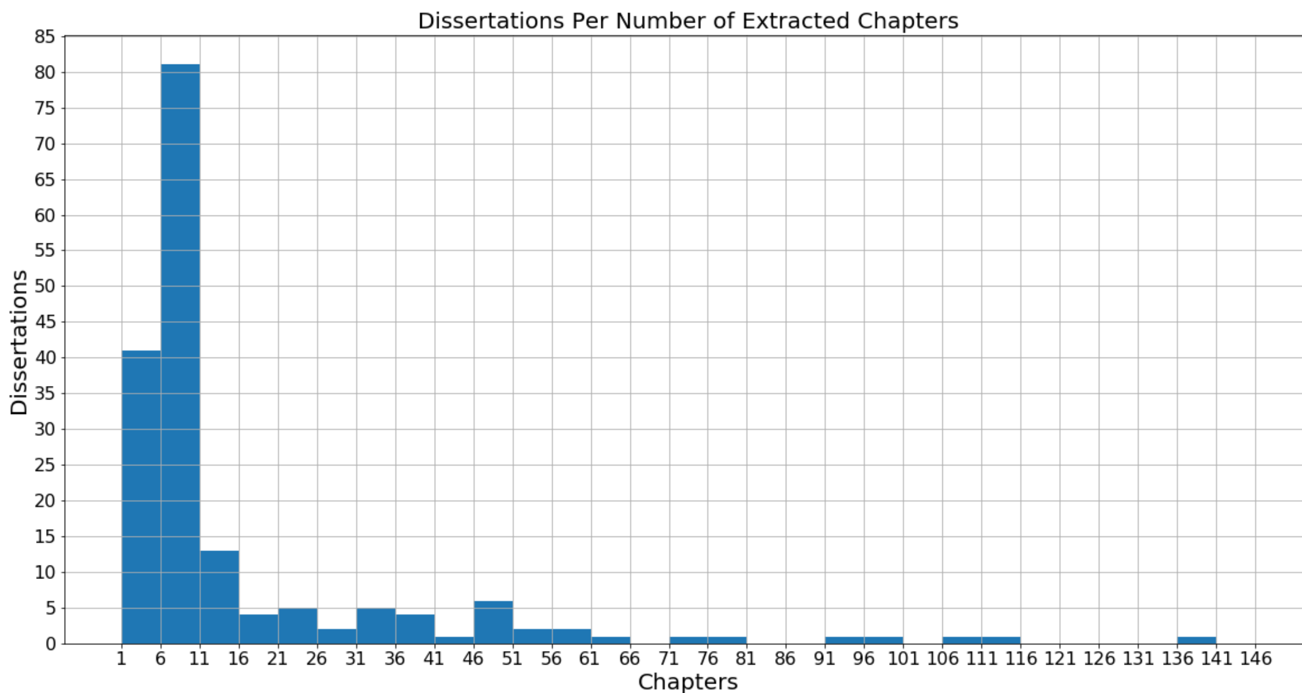


Figure 21: Dissertations Per Number of Extracted Chapters

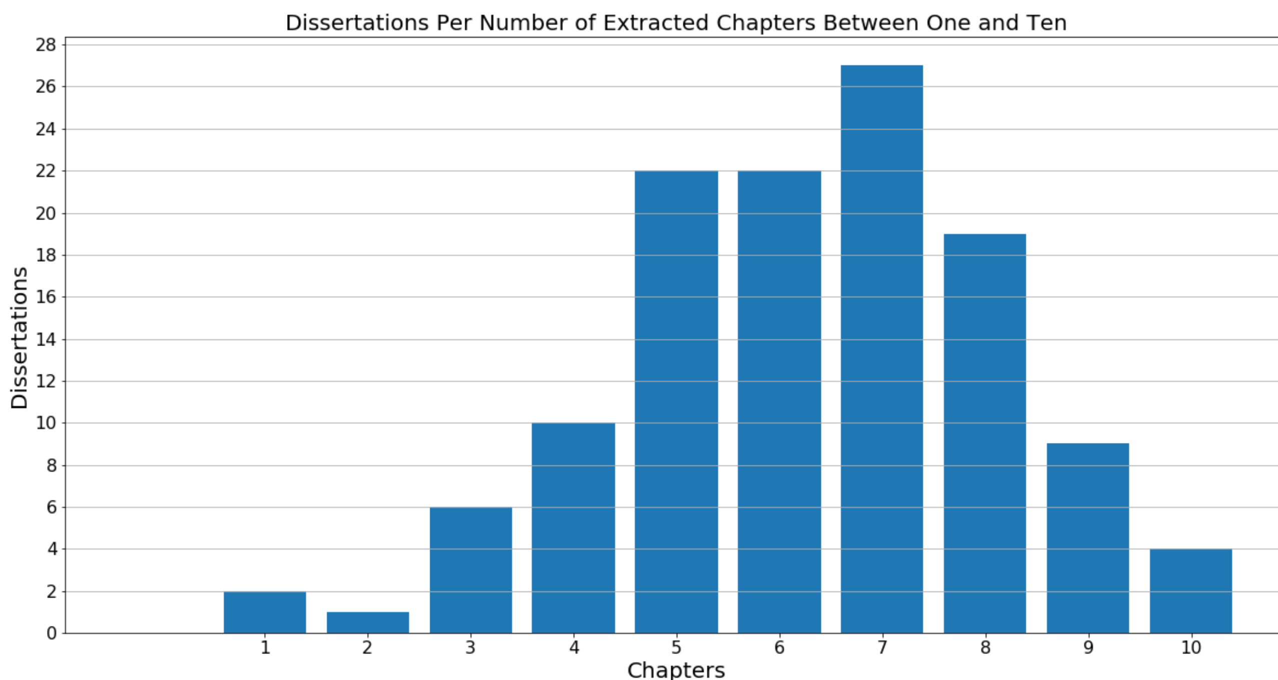


Figure 22: Dissertations Per Number of Extracted Chapters Between One and Ten

From Figure 21, we can see that the number of chapters range from 1 to 136. Of the 174 processed dissertations, approximately 75% are between 1 and 10 chapters. Figure 22 shows the exact number of dissertations within that range. From Jude's reasoning for choosing ITCore, we can confirm that ITCore is performing relatively well. Further testing using the previously mentioned metrics is necessary.

6.2 Classification

6.2.1 Qualitative Evaluation

To get a better understanding of the underlying term distribution within our collection, we used word clouds to help visualize terms based on their TF-IDF ranks. We used the Python `wordcloud` package to create word clouds from a subset of metadata fields from the dissertation and thesis subsets. We start by analyzing the abstracts and titles from the handle.txt files. Comparing Figures 23 and 25 to Figures 24 and 26, we noticed that both the titles and abstracts of these theses and dissertations are similar. The abstract word cloud however seems to have more noise compared to the word cloud of titles. You can notice contextually less relevant terms (e.g., "two", "may", "new", and "thesis") in Figure 25 for thesis abstracts, which are not prevalent in the thesis title word cloud in Figure 26. That is expected, however, as abstracts are more descriptive than titles.

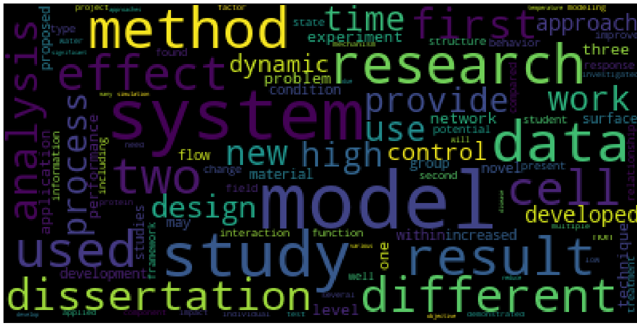


Figure 23: Word cloud of abstracts
(Dissertation Subset)

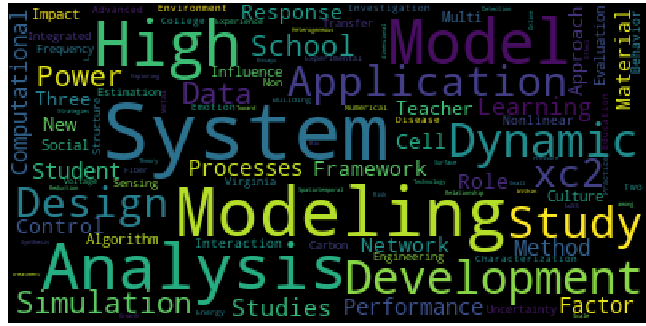


Figure 24: Word cloud of titles
(Dissertation Subset)

Figure 25: Word cloud of abstracts
(Thesis Subset)

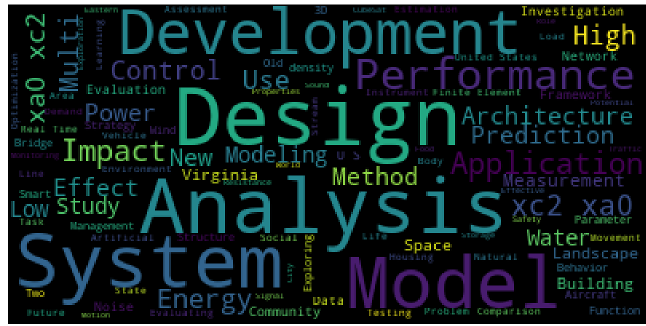


Figure 26: Word cloud of titles
(Thesis Subset)

Next we generated word clouds for the author department metadata and subjects generated by our classification model. Since the abstracts and titles did not have any distinguished subject related keywords in the word clouds, we decided to compare just the author department metadata to the subjects generated by our classification model. One interesting observation is the higher diversity in author departments for theses than dissertations. This aligns with Jude's observation in her exploratory data analysis. While master's theses were generally dominated by STEM related disciplines, there was a high number of theses from non-STEM related disciplines. However, doctorate dissertations had an even more skewed distribution favoring STEM related works and thus reduced discipline variation [17].

Comparing Figures 27 and 29 to Figures 28 and 30, we noticed that the subjects were more granular than the author departments. Since the subjects have access to the fulltext and are based on subjects determined from ProQuest's subject categories, the increased granularity is expected. At the same time we also noticed a few similarities in the subjects identified by the classifier and those present in the author departments, giving confidence in our classifier.

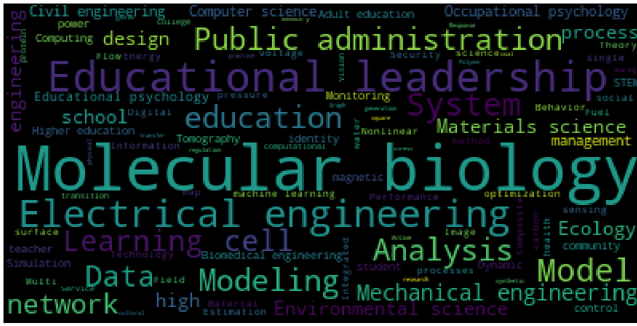


Figure 27: Word cloud of subjects (Dissertation Subset)

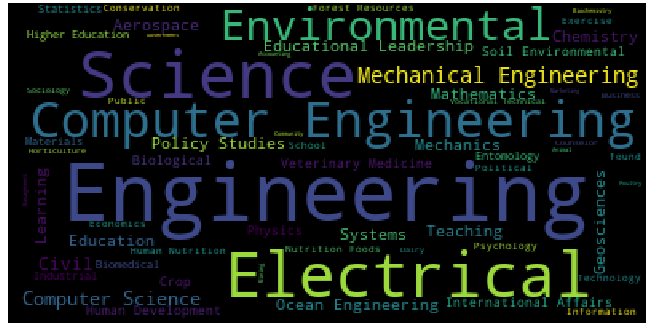


Figure 28: Word cloud of author departments (Dissertation Subset)

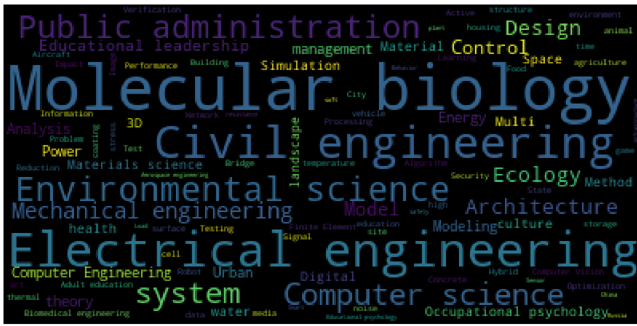


Figure 29: Word cloud of subjects (Thesis Subset)

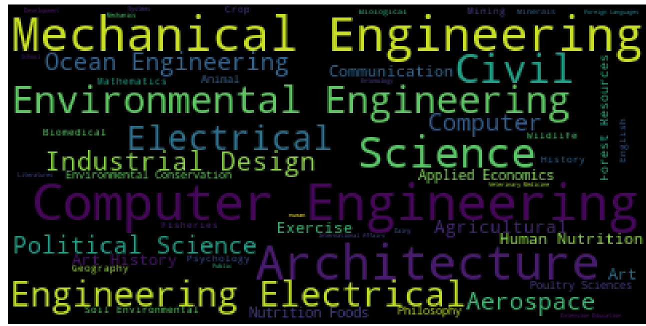


Figure 30: Word cloud of author departments (Thesis Subset)

Qualitative evaluation can gives us a general idea of how our classifier works and how the existing metadata might contribute to the performance. But, qualitative evaluation is not a good way to decide whether a classifier is accurate. Thus in the next section we talk about some quantitative evaluations.

6.2.2 Quantitative Evaluation

The four main metrics for classification that we are using include: accuracy, precision, recall, and F1-score. Each of these metrics is derived from a function on true positives (TP), true negatives (TN), false positives (FP) and/or false negatives (FN). TP, TN, FP and FN can be represented as a confusion matrix like the one shown in Figure 31.

In the context of document multi-label classification, we have TP, TN, FP, and FN values for each possible label as compared to the rest of the labels. For example, if we are considering "Statistics" as the label at hand, the true positives represent all of the test documents that were classified correctly to be under the Statistics Department. The true negatives represent all the test documents that were correctly classified to not be "Statistics". The false positives are all the test documents that do not belong in the Statistics Department but were incorrectly classified to belong in the Statistics Department. The false negatives are all the test documents that do belong in the Statistics Department but were incorrectly classified to belong in a different department. Given

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	True Positive	False Negative
Class 2 Actual	False Postive	True Negative

Figure 31: Confusion Matrix

these definitions of TP, TN, FP, and FN we can define accuracy, precision, recall, and F1-score using the following equations:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \qquad \text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{F1 score} = \frac{2 * Recall * Precision}{Recall + Precision}$$

To find the overall accuracy, precision, recall, and F1-score we take the average over all labels. Note: All of the above metrics are a fraction ranging from 0 to 1.

Preliminary Results: Jude's Method By running the test ProQuest dataset through the classification pipeline we get the average accuracy, precision, recall and F1-score as shown in Table 4.

Algorithm	Accuracy	Precision	Recall	F1-score
Random Forest	40.2	55	41.5	46.4
Support Vector Machine	50.6	72.4	56.6	63

Table 4: ProQuest Test ETD Evaluation

From the results in Table 4, we can see that the Support Vector Machine model performs better on average, but specifically it performs better with precision, which suggests that our model is better at differentiating true positives and false positives than it is differentiating true negatives and false negatives. This is an important factor to consider as we make further improvements to the classification model.

Figures 32 and 33 show the best and worst models among all the models we generated. Additionally, the figure displays the hyper-parameters used for the Doc2Vec embedding. SVM performed

SVM

D2V Emebbedings: 0dm_8win_200dim_50e
fastText: 100emodel3N100D

Accuracy: 0.5655526992287918
Precision: 0.760351473325713
Recall: 0.6149068322981367
F1-score: 0.6742543200452709

Figure 32: Best Model Metrics

RF_D

D2V Emebbedings: 0dm_8win_200dim_50e
fastText: 100emodel3N100D

Accuracy: 0.39203084832904883
Precision: 0.5412665414161566
Recall: 0.4101444407121263
F1-score: 0.45723574056429284

Figure 33: Worst Model Metrics

the best with an accuracy of 56.6% accuracy. Even the best model we generated had a relatively low metric score, which left much to be desired with model accuracy.

Once we had generated our classifiers and their metrics, we used the best model and created a pipeline to classify a subset of the Virginia Tech ETDs within our IR storage. However, we can not measure any metrics since Virginia Tech's ETD department categories don't match with the PQDT category labels we trained our classifiers on. Only 6 of the 48 departments in our ETD subset exactly match with the ProQuest subject categories. There are a few subjects such as Aerospace Engineering and Industrial Engineering within the ProQuest categories that are similar to the department names found in our ETDs: Aerospace and Ocean Engineering, and Industrial and Systems Engineering, respectively. Such differences call for either a model trained on just our ETDs or preprocessing of our department names to align with ProQuest's subject names for the known entries.

Preliminary Results: Probabilistic ETD Classification After running the ProQuest dataset through the new pipeline, we generated metrics as described in Figure 34. The metric ratings for the second pipeline are higher compared to the first pipeline we created using Jude's method.

Since we haven't experimented with varying any of the hyper-parameters in the second pipeline,

```
Accuracy: 0.6362467866323908
Precision: 0.6687867248785252
Recall: 0.6362467866323908
F1-score: 0.6185095950845124
```

Figure 34: Second Pipeline Metrics

```
{
    "word_segments" : 1000,
    "method" : "cumalative",
    "C" : 1.25,
    "ngram_range" : (1, 3)
}
```

Figure 35: Second Pipeline Parameters

there is also a high chance we can get better metrics by attempting to fine-tune the hyper-parameters.

6.3 Image Detection

For evaluating the image detection, we focused on categorizing cases when the image detection fails.

Figure 36 shows an example where the microservice correctly identified a figure, but additionally identified a subsection as a separate figure. It is possible to mitigate these errors in the cropping service by opting not to crop boxes completely contained by other boxes.

Another example of failure is the case where a section like the table of contents is labeled as a figure. Figure 37 shows such a case. Note that saying whether or not this can undeniably be called a failure can be debated. For our evaluation we do not consider this to be a figure, however there may be some use cases where extracting the table of contents can be useful.

bundles; cells 1, 2, and 6 have maximum tensions of 100 pN or higher, while cells 3, 4, and 5 have maximum tensions of less than 40 pN.

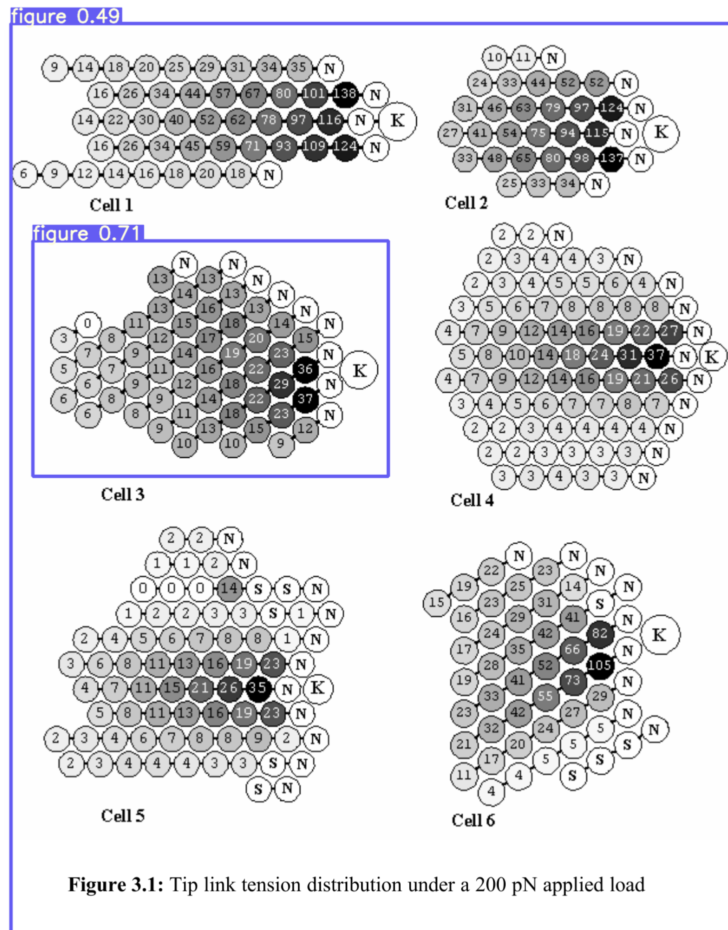


Figure 3.1: Tip link tension distribution under a 200 pN applied load

Figure 3.2 shows information about the bundles' stiffness. Part I of the figure shows the bundle stiffnesses under a 500 pN applied load. Note that the stiffnesses of cells 3, 4, and 5 are an order of magnitude higher than those of cells 1, 2, and 6. The

CHAPTER 3: THREE DIMENSIONAL BUNDLE MECHANICS 23

Figure 36: A case where the image detection microservice incorrectly labeled part of a figure

6.4 ETD Pipeline Evaluation

Although we were able to get our microservices deployed onto Airflow, and collaborated with the INT team to create a corresponding workflow, we were unable to bring the pipeline to full fruition. The main roadblock stems from trouble getting one of our services to communicate with a MySQL database running in the cluster. This section describes the specific challenges that prevented us

figure 0.85

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
TABLE OF CONTENTS iv
LIST OF FIGURES vi
LIST OF TABLES vii
CHAPTER 1: INTRODUCTION AND BACKGROUND 1
The Semicircular Canals 1
The Otolith Organs 1
Hair Cells: Anatomy 2
Hair Cells: Physiology 6
Directional Sensitivity 6
Adaptation 7
Other theories 8
Mechanical testing of bundles 9
Motivation for Modeling 9
Previous Research Work and Modeling Assumptions 10
Summary of thesis research 11
Organization of the thesis 12
CHAPTER 2: METHODS AND MATERIALS 13
Model Features 13
Model Limitations 15
Modifications 16
CHAPTER 3: THREE-DIMENSIONAL BUNDLE MECHANICS 22
Procedure 22
Tip Link Tensions and Bundle Stiffness 22
Results 22
Discussion 25
Effect of Cilia Diameters 29
Effect of Increased Tip Link Modulus 30
2-D vs. 3-D Comparison 35
Effect of Distributed Load 38
Summary/Conclusions 39
CHAPTER 4: ION GATES 41
Tip Link Ion Gates 41
Procedure 41
Results 42
Discussion 45

TABLE OF CONTENTS iv

Figure 37: Table of Contents labeled as an image

from completing this goal. It also describes accomplishments according to some metrics, of the work completed to the best of our abilities.

6.4.1 Timing

Although we are unable to time the pipeline on Airflow, we are aware that the speed of some of the services could be improved. The classification microservice takes the longest to complete; the image_inference and the chapter_extraction microservices also take a notable amount of time to run. These microservices take so long because they either have to load models or, in the case of ITCore, run its own rule-based processing. Going forward the future team should improve upon the regression tests to provide a better estimate for runtime and scalability. The time to load models could be improved by offloading the processing to an API. This would solve the issues of loading models for every pipeline since the API would be persistent. However, at scale, this may result in the API becoming a bottleneck.

6.4.2 Pipeline Status

- All services except for chapter_extraction have been unit tested using the GitLab Runner and deployed to the respective repository registry.
- Each service within the pipeline has been unit tested on the subset of 961 theses and dissertations created by a previous CS5604 team.
- We have not been able to process the main collection because of issues faced trying to successfully deploy the chapter_extraction container in Airflow. The main issue with deploying the chapter_extraction container was both trying to install the dependencies needed for MySQL's CLI and trying to insert tables into a MySQL instance from the chapter_extraction container. In our attempts to correct this issue, we have tried multiple base images (e.g., `Ubuntu:18.04`, `mysql:latest`, `python:3.8-slim-buster`) with no clear and consistent build solution. The closest we have gotten to a successful and consistent deployment is by building from an `Ubuntu:18.04` image, installing the necessary keys for `mysql-client`, and then installing `mysql-client` via `apt-get`[4]. The problem with this method is that the command to get the necessary keys will sometimes work and sometimes fail. We have not been able to identify a cause for this behavior.

[4] The Dockerfile with these build steps can be found in the chapter_extraction repository.

7 User Manual

The services created consist of microservices and APIs. The microservices are designed for use in Airflow, while the APIs are designed for individual use. In this section we will first cover the microservices, then the APIs.[5]

7.1 Microservices

This section describes the input and output for each microservice. The input and output supplied to these microservices is controlled via environment variables, however how one specifies those variables depends on where the microservice is located. If you are running the pipeline as part of the CS5604 Airflow pipeline, input and output is controlled through an online dashboard created by the FE team.

7.1.1 Validation

Validation consumes the metadata and populates the Postgres service.

> **Input:** A file path indicating the location of the target PDF file on NFS, a file path indicating the location of the JSON file with the handle ID, a file path indicating the location of the JSON file with the document type and a file path indicating the location of the text file with the existing document metadata.
>
> **Output:** A file path to the resulting validated file which indicates if validation completed successfully.

7.1.2 Figure and Table Extraction

Figure and Table extraction is broken up into 3 microservices.

PDF To Images

> **Input:** A file path indicating the location of the target PDF file on NFS.
>
> **Output:** A file path indicating a directory that contains one or more PNG files on NFS. Each PNG is a picture of one page from the input PDF file.

Figure Inference

> **Input:** A file path indicating a directory on NFS that contains one or more PNG files.
>
> **Output:** A file path indicating a directory on NFS that contains bounding boxes for any figures discovered in the input PNG files and a copy of each input PNG.

[5] Some of the examples for accessing the endpoints use a tool called HTTPie [36]. HTTPie is used for the same purpose as cURL, except the syntax is debateably easier.

Crop Images

Input: A file path indicating a directory on NFS that contains one or more PNG files and one or more file of bounding boxes that will be extracted from their corresponding PNG files.

Output: A file path indicating a directory on NFS that contains the images that result from cropping the input bounding boxes from the input PNG files.

These services can also be linked together in the form of a pipeline using Apache Airflow.

7.1.3 Chapter Extraction

Input: A file path indicating an ETD on NFS.

Output: A file path indicating a directory on NFS that will be used to store the chapter PDF files.

7.1.4 Fulltext Extraction

Input: A file path indicating an ETD PDF file or Chapters PDF file directory on NFS.

Output: A file path indicating a directory on NFS that will be used to store the Chapter Texts for the Chapter PDF files and a file path indicating a file on NFS that will be used to store the ETD Text.

7.1.5 Classification Using Probabilistic ETD Classification

This classification service is also part of the entire ETD pipeline and exists as a microservice. The microservice classifies an ETD and its segmented chapters using environment variables.

Input: A file path indicating an ETD Text file or Chapter Text directory on NFS and a file path to the Dublin Core XML file.

Output: A file path to the resulting classification file which indicates if classification completed successfully.

7.1.6 Ingestion

Ingests and indexes the metadata from Postgres into Elasticsearch.

Input: A file path to the finish classification file which indicates if classification completed successfully.

Output: A file path to the resulting pipeline file which indicates if the pipeline completed successfully.

7.2 APIs

The following APIs are artifacts from development over the semester. These APIs can be useful for individual exploration. Some of the APIs also allow batch processing. The probabilistic classification API has multiple models with different parameters. While these APIs are not used in the final workflow, they may serve as useful starting points for other methods of implementation or new research goals.

7.2.1 Chapter Extraction

This service supports the extraction of chapters from a given ETD.

/extract/ The extract endpoint allows for the extraction of chapters of an ETD.

```
http POST <domain>/extract/chapters/<pdf_uri>
```

Performs chapter extraction on a given ETD.

7.2.2 Classification Using Jude's Method

This classification API uses Jude's work (see Section 4.2.2) and allows users to generate an ETD corpus's category labels.

/preprocess Endpoint that allows for the preparation of an ETD. It returns the ETD preprocessed in dictionary form.

```
http GET <domain>/preprocess
```

Returns the endpoint's welcome message.

```
curl -F dublincore=@<dublincore_file_address> \
   -F pdf=@<ETD_PDF_address> domain/preprocess
```

Returns a dictionary of an ETD prepossessed.

/classifytext Endpoint that supports the classification of a corpus of ETD(s).

```
http GET <domain>/classify
```

Returns the endpoint's welcome message.

```
curl -d@<JSON_file> -H 'Content-Type:  application/JSON' domain/classifytext
```

Returns a dict with the list of predictions for each ETD.

7.2.3 Classification Using Probabilistic ETD Classification

This classification API uses probabilistic ETD classification (see Section 4.2.3) and allows users to generate an ETD corpus's category labels.

/preprocessetd Endpoint that allows for the preparation of an ETD. It returns the ETD preprocessed in dictionary form.

```
http GET <domain>/preprocessetd
```
Returns the endpoint's welcome message.

```
curl -F 'metadata=@<xml_file>' \
   -F 'etdDocuments=@<pdf_or_text_file>' domain/preprocessetd
```
Returns a dictionary of an ETD prepossessed.

/classifyetd Endpoint that supports the classification of a single ETD.

```
http GET <domain>/classify
```
Returns the endpoint's welcome message.

```
curl -F 'metadata=@<xml_file>' \
   -F 'etdDocuments=@<pdf_or_text_file>'
   -F 'isChapter=<True/False>' -F 'isPDF=<True/False>' domain/classifyetd
```
Returns a dict with ETD handle as key and generated categories as a list.

/classifycorpus The classification endpoint that supports the classification of a corpus of ETD(s).

```
http GET <domain>/classify
```
Returns the endpoint's welcome message.

```
curl -d@<JSON_file> -H 'Content-Type:  application/JSON' domain/classifytext
```
Returns a dict with each ETD(s) handle's as key's and their generated categories.

7.2.4 Elasticsearch Management

The purpose of this service is to be the liaison between the other ETD services and the CS5604 Elasticsearch instance. In most cases, this service will preprocess data where needed, then send a request to Elasticsearch; the response from Elasticsearch will be passed back to the client. This service has 2 main endpoints:

/indices The indices endpoint allows for the creation, modification, and deletion of endpoints.

```
http GET <domain>/indices
```
Returns a list of all indices present in Elasticsearch.

```
http POST <domain>/indices/<index_name> < <index_mapping>
```

Adds an index called <index_name> into Elasticsearch. The body of the request is a JSON map of an Elasticsearch index.

`http PUT <domain>/indices/<index_name> < <index_mapping>`

Modifies an existing index called <index_name>. The body of the request is a JSON object with the changes that need to be applied to the index.

`http DELETE <domain>/indices/<index_name>`

Deletes an index called <index_name>.

/**index** The index endpoint allows for addition of new records.

`http POST <domain>/index/<index_name> < <data>`

Adds records to an index.

8 Developer Manual

8.1 Accessing ETDs

Ceph The collection of ETDs is mounted onto a server accessible via ir.cs.vt.edu. To access the data perform the following tasks:

- Login into the Virginia Tech VPN (Pulse). The server is only accessible if you are using the VPN.
- SSH into the server (e.g. ssh <pid>@ir.cs.vt.edu) where <pid> is your Virginia Tech PID.

The collection of the data is in /mnt/ceph/cme. The main folders of interest are dissertation/, dissertation_subset/, thesis/, and thesis_subset/. These folders contain either an entire collection or a subset of the collection. Each document has its own folder and each folder is labeled as a document's handle. An example of the contents of each folder is shown in Figure 38.

```
[        @ir 37478]$ ls -l
total 909
-rw-r--r-- 1 root root   4105 Oct  3  2019 37478.txt
-rw-r--r-- 1 root root 703227 Oct  3  2019 Knepp_KA_D_2012.pdf
-rw-r--r-- 1 root root   1824 Oct  3  2019 Knepp_KA_D_2012.pdf.jpg
-rw-r--r-- 1 root root 209445 Oct  3  2019 Knepp_KA_D_2012.pdf.txt
-rw-r--r-- 1 root root    172 Oct  3  2019 contents
-rw-r--r-- 1 root root   8292 Oct  3  2019 dublin_core.xml
-rw-r--r-- 1 root root     12 Oct  3  2019 handle
-rw-r--r-- 1 root root    410 Oct  3  2019 metadata_thesis.xml
[        @ir 37478]$ |
```

Figure 38: Contents of a thesis folder

These contents include the scraping and extraction work of prior teams:

- ***handle*.txt** contains the manually generated metadata formed by combining the metadata thesis file with the Dublin Core metadata.
- ***name*.pdf** contains the ETD PDF of interest.
- ***name*.pdf.jpg** contains the image version of the ETD PDF or in some cases the scanned version of the ETD PDF.
- ***name*.pdf.txt** contains the fulltext version of the PDF file from the library's efforts using OCR.
- **contents** describes what is in the folder.

- **dublin_core.xml** contains metadata in the Dublin Core Metadata Initiative (DCMI) format. You can read more about the specification and format at their website: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/.
- **handle** contains the handle number which is a unique identifier for the document. This value can also be used to build the URL which helps retrieve the document from VTechWorks.
- **metadata_thesis.xml** was generated by the Virginia Tech library. Contact the library staff if the need for more details arises.

NFS All newly generated data will be stored in NFS. Instead of each folder labeled by the document's handle, it is labelled by the run ID (also known as workflow ID). You can find the association between handle ID and workflow ID from our Postgres Database described below.

Postgres The Postgres database is used to store and update metadata as we permute through each service in the pipeline. All of the Postgres data is stored in the ETD database found as a container hosted on `cloud.cs.vt.edu` under the CS 5604 ETD 2020 – testing cluster. The ETD database has a metadata scheme which hosts two tables: `metadata.etd` (Table 5) and `metadata.chapter` (Table 6).

Table 5: Postgres Metadata Schema

Column Name	Column Type	Column Description	Service
index	bigint	Auto generated Unique identifier	AUTO GENERATED
contributor_author	text	main author of the document	validate_input
contributor_committeechair	text	chair(s) of the author's advisory committee	validate_input
contributor_committeecochair	text	co-chair(s) of the author's advisory committee	validate_input
contributor_committeemember	text	member(s) of the author's advisory committee	validate_input
contributor_department	text	the major/department that the author belongs to	validate_input
date_accessioned	text	date the document was added to the collection	validate_input
data_available	text	date the document became available for viewing	validate_input
date_issued	text	date of formal issuance	validate_input
date_adate	text	unknown	validate_input
date_sdate	text	unknown	validate_input
date_rdate	text	unknown	validate_input
continued ...			

... continued			
Column Name	**Column Type**	**Column Description**	**Service**
degree_discipline	text	degree awarded	validate_input
degree_grantor	text	entity that granted the degree	validate_input
degree_level	text	level of degree; usually "doctoral" or "masters"	validate_input
degree_name	text	name of degree; usually "PHD" or "MS"	validate_input
description_abstract	text	a document's abstract	validate_input
description_degree	text	degree; either "MS" or "PHD"	validate_input
description _provenance	text	any changes to a documents ownership that are noteworthy	validate_input
description _sponsorship	text	any notes on entities who have sponsored the research	validate_input
format_medium	text	type of document; always "ETD"	validate_input
handle	bigint (NOT NULL)	unique identifier for documents	validate_input
identifier_other	text	unique identifiers for the document that are not the handle	validate_input
identifier_uri	text	a link to the actual document	validate _input
publisher	text	entity responsible for making a resource available	validate_input
relation	text	handle(s) of related document(s)	validate_input
relation_haspart	text	files that make up the original document	validate_input
rights	text	any copyright or other ownership information	validate_input
subject	text	a list of subject/keywords describing the document	validate_input
subject_lcc	text	Library of Congress categories	validate_input
subject_lcsh	text	Library of Congress subject headers	validate_input
title	text	the title of the document	validate_input
etd_filename	text	filename of ETD PDF	validate_input
etd_uri	text	filepath of ETD PDF	validate_input
figures_folder _uri	text	directory path of the figures extracted	crop_image
figures_total _count	bigint	total count of the figures extracted for a ETD	crop_image
tables_folder _uri	text	directory path of the tables extracted	crop_image
tables_total _count	bigint	total count of the tables extracted for a ETD	crop_image
updated_bool	boolean	indicates if the metadata has been updated post ingestion into elastic search	validate_input
continued ...			

Column Name	Column Type	Column Description	Service
			...continued
last_update _ts	timestamp	indicates the date and time of last update	validate_input
text_uri	text	filepath of ETD Text	etd_text _extraction
workflow_id	text	airflow run id	validate_input
doc_type	text	Dissertation or Thesis	validate_input
etdhandlepk	(handle, doc_type)	unique identifier formed by combining handle number and document type	N/A

Table 6: Postgres Chapters Schema

Column Name	Column Type	Column Description	Service
index	bigint	Auto-generated unique identifier	AUTO GENERATED
handle	bigint	ETD unique identifier	chapter_extraction
doc_type	text	document type, either "Dissertation" or "Thesis"	chapter_extraction
chapter_number	bigint	Specifies order of chapters	chapter_extraction
chapter_uri	text	Filepath to chapter	chapter_extraction
subject	character	Subject labels for the chapter text as predicted by the classifier	etd_probabilistic _classification
chapter_fulltext _uri	text	Text extracted from chapter	etd_text_extraction
pages	text	Start and end page number of the chapter	chapter_extraction
page_number	bigint	Chapter filename as well as start page number of chapter	chapter_extraction
updated_bool	boolean	Indicates if the metadata has been updated post ingestion into elastic search	ANY
last_update_ts	timestamp	Indicates the date and time of last update	ANY
handlepagenum compositekey	PRIMARY KEY (handle, page_number)	Unique identifier formed by combining handle number and page number	N/A
etdhandlefk	FOREIGN KEY (handle, doc_type)	Unique identifier tying the etd table and chapter table formed by combining handle number and document type	N/A

8.2 Git Repositories for Services

The repositories for our pipeline and our APIs can be found at the Gitlab group page: https://git.cs.vt.edu/cs-5604-fall-2020/etd. Each repository corresponds to a single microservice. Some repositories also contain an API branch that contains a Flask app with capabilities similar to

the microservice. Additional details not mentioned in this report can be found in the `README.md` of each repository.

8.2.1 Microservices

Directory Structure The top level of each repo contains a `.gitlab-ci.yml` file that defines the testing pipeline for the Gitlab Runner. Each of the services follows a similar layout.

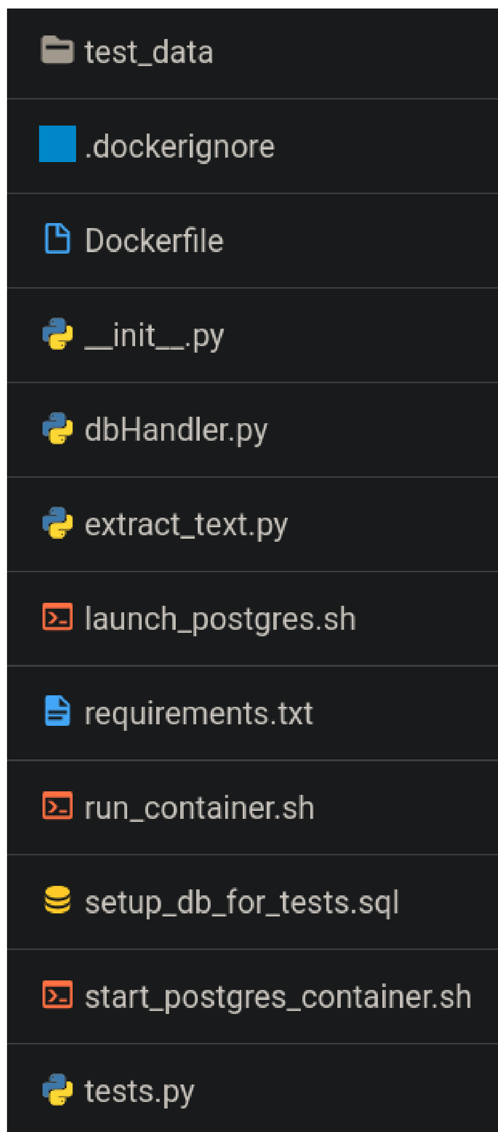


Figure 39: Text Extraction service directory structure

Every service contains a `Dockerfile`, `.dockerignore`, `test_data` folder, `requirements.txt`, test scripts, and a main script. Some services, like the service shown in Figure 39, contain extra scripts in order to launch and connect to a Postgres database. `dbHandler.py` is a module built by the INT team to streamline connections to the Postgres database. The `.sh` files are used to easily configure and spawn the microservice and Postgres containers. Any `.sql` files present in the

directory (e.g., `setup_db_for_tests.sql`) are used in the `.sh` files and in the Gitlab runner in order to populate the Postgres database for testing.

The pytest files in each repository contain regression tests. The `test_data` folder holds data for testing the services. See the individual tests for more information on how the data is used.

Running Services Each microservice can be built with `docker build`. `run_container.sh` contains a Docker run command; the command will need to be modified to fit your environment. The input and output can specified using the `-e` flag with the `docker run` command. As an example, the classification microservice can be run with the following command:

```
docker
   -e HANDLE_ID=/mnt/handle_id.json
   -e DOC_TYPE=/mnt/doc_type.json
   -e ETD_TEXT=/mnt/etd_text
   -e DUBLIN_CORE=/mnt/dublin_core.xml
   -e CHAPTERS_TEXT=/mnt/chapter_text
   -e POSTGRES_CONN_STRING=database_connection_string
   -v /your/local/folder:/mnt
   -name etd_classification
   etd_classification:latest
```

Launching Postgres Services that talk to the Postgres database contain two scripts related to Postgres: `start_postgres_container.sh` and `launch_postgres.sh` The former only starts the Docker container while the latter both starts the container and populates the database using the `.sql` script in the same directory.

The following subsections contain extra per-service instructions and information.

8.2.2 Chapter Extraction

For our Chapter Extraction we build on the ITCore package available on Github (https://github.com/intextbooks/ITCore,[5]). ITCore is shared as a Maven (i.e., a build automation tool for Java projects) project. To process your data set using ITCore you need to follow the steps below:

1. Clone ITCore from Gitlab (https://git.cs.vt.edu/cs-5604-fall-2020/etd/full_etd_pipeline/services/chapter_extraction)

2. Download MySQL from https://dev.mysql.com/downloads/mysql/ or run a MySQL Docker container from the latest MySQL image and expose port 3306 as follows:
`docker run -d -p 3306:3306 -name=mysql-test mysql`

3. Either download Eclipse and the Maven plugin for the IDE or just Maven.

4. Once downloaded, open MySQL Workbench and create the intextbooks_db database by running the `intextbook_db.sql` in the main src folder OR if using a Docker container to initialize the database within Docker.

5. If using Eclipse with the Maven plugin, import the ITCore code as a Maven repository. Otherwise, follow this "Maven in 5 Minutes" tutorial: http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html to download the dependencies from the pom.xml file found in the base ITCore folder.

6. Next, navigate to the `ProcessExample.java` file found under `src/examples` in the base ITCore folder. Run the Java file either through Eclipse or command line, passing in the input PDF file and desired output folder as arguments. Note: Make sure your filepath does not contain any whitespaces as ITCore does not handle whitespaces in filepaths well.

7. Upon navigating to the desired output directory you passed in as an argument, you will see the following folder structure in Figure 40.

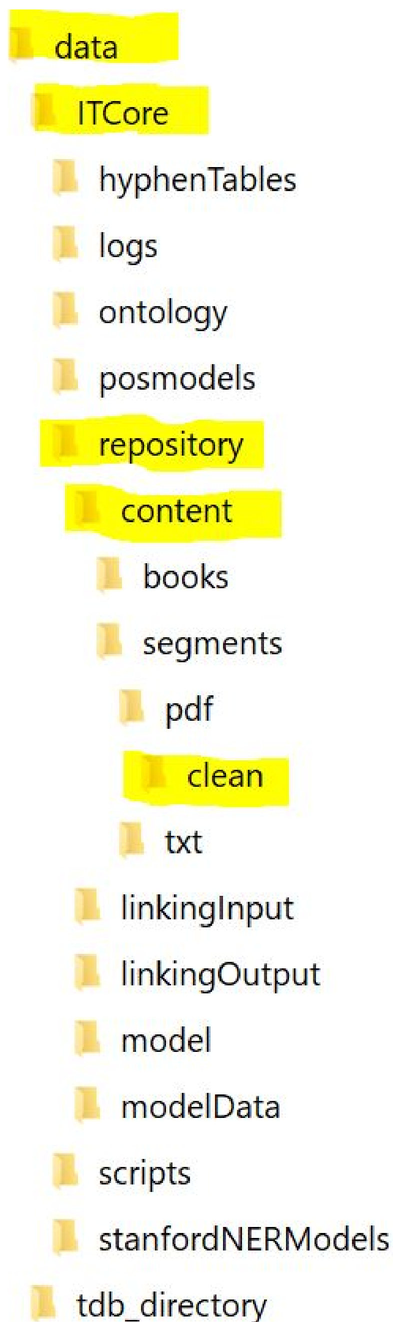


Figure 40: Structure of output directory upon running ITCore

The clean folder will have all of your segmented PDF files, with a PDF file for each chapter named by the starting page of the chapter.

To run a batch of PDF files, use the bash script named `run_seg.sh` after updating the class path and input/output directories defined in the script. You can find the class path using either Maven or Eclipse.

8.2.3 Figure Extraction

Our figure extraction services are based on the work of Kahu and others. Please refer to the README [18] for information on running that full pipeline and using features that we were unable to port like training. The following is a brief tour of each of the figure extraction services:

8.2.4 PDF To Images

`pdf_to_image.py`: This small Python module contains the main functionality of this service. A single function is used to transform a PDF into a collection of PNG files.

`/test_data`: This directory contains the PDF files that will be used to run CI/CD tests.

8.2.5 Figure Inference

`detect.py`: This is the main module that gets called to run inference on images.

`inference_tests.py`: This is the pytest file that gets run in the CI/CD system. "test.py" is a file from YOLOv5 that we are not utilizing.

`run_detection.sh`: This is the script that gets run by the Docker container. It calls "detect.py" with the correct parameters and specifies the paths to the weights that will be used by the inference model. If you would like to use different weights for inference, modify the parameters here or modify the current weights file.

8.2.6 Crop Images

`crop_image.py`: This small Python module contains the main functionality of this service. A single function is used to crop images from bounding boxes.

8.2.7 Text Extraction

`extract_text.py`: In this service, this is a Python module that contains the main function of transforming PDF file to text file.

8.2.8 Classification

The following briefly describes important files and folders within the classification microservice:

1. `pretrainedModels/`: Directory of pretrained logistic regression models for each classification category.

2. `pretrainedVectorizers/`: Directory of pretrained TF-IDF vectorizers for each classification class.

3. `classifyETD`: Main file. Loads the pretrained models and vectorizer. It uses the environment variables to classify an ETD and its segmented chapters.

4. classify_utils: Contains all classification routines needed such as segmenting ETD(s) and generating cumulative probabilities for each segment.

5. support_utils: Contains supporting routines such as preprocessing text and extracting metadata.

As mentioned above, pretainedModels and pretrainedVectorizer are loaded within classifyETD.py to classify each ETD. This is inefficient as the service needs to reload the models and vectorizers each time it needs to classify a new ETD. Something that could boost the speed of the microservice is to containerize the models and vectorizers so they are always loaded in memory. Follow this by modifications to classifyETD.py to communicate with the container to speed up overall classification efficiency.

8.3 CI/CD

8.3.1 Unit Tests

Each repository has regression tests defined in either a Python file named tests.py or a directory named tests/. All of the tests are defined using the pytest framework. The test coverage varies across repositories, but they mainly consist of error handling tests and correct output tests.

8.3.2 Gitlab Runner

Each microservice repository contains a .gitlab-ci.yml that defines a testing pipeline for the Gitlab Runner. An example is shown in Figure 41. Each microservice's workflow is divided into two stages: test and build. The test stage installs all of the dependencies and executes the unit tests specified in the repository. The build stage attempts to build the microservice as a Docker image; if the pipeline is run for master branch, the successfully built container will be automatically pushed to the repository's container registry.

Though most of the microservices run into no issues when building, the chapter_extraction container has been inconsistent in when it fails and when it succeeds. This inconsistency is caused by the dependencies that need to be installed to run the service, specifically mysql-client. mysql-client is necessary in order to communicate with the MySQL container defined in the test environment. The inconsistency is caused by downloading the MySQL GPG public key. This key is necessary to download mysql-client. Sometimes, this command fails; the behavior is inconsistent and we have not found a better alternative. Other methods for downloading MySQL that are provided by MySQL themselves have also resulted in similar inconsistencies or total roadblocks. As a result, we have tested and built the container locally before pushing it to the package registry.

8.3.3 Deploy Tokens

In order to deploy containers to Airflow, Airflow needs read access to the ETD containers. Access to the containers is given via deploy tokens. These deploy tokens can be managed in Settings > Repository > Deploy Tokens. An important note is that all of the deploy tokens have expiration dates; once expired, Airflow will require new tokens in order to access the containers. The current token are set to expire in ten years.

```yaml
variables:
    DOCKER_DRIVER: overlay2
    DOCKER_TLS_CERTDIR: ""
    PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"
    ETD_TEST: "true"
    # POSTGRES VARIABLES
    POSTGRES_USER: "etd"
    POSTGRES_PASSWORD: "etd"
    POSTGRES_DB: "etd"
    PGPASSWORD: "etd"
    POSTGRES_CONNECTION_STRING: "postgres+psycopg2://etd:etd@postgres:5432/etd"

stages:
  - test
  - build

test:
    stage: test
    image: python:3.8-slim
    services:
        - postgres:13
    script:
        - apt-get update && apt-get upgrade -y && apt-get install -y ffmpeg libsm6 libxext6 postgresql postgresql-contrib
        - psql -U etd -h postgres -d etd < setup_db_for_tests.sql
        - pip install -r requirements.txt
        - echo "Running tests"
        - pytest tests.py

docker-test-build:
  # If not on master, just check that it builds properly
  stage: build
  script:
    - echo "Building container"
    - docker build -t etd_crop_image:latest .
  except:
    - master

docker-test-and-deploy:
  # If running from master, build and deploy
  image: docker:latest
  stage: build
  services:
    - docker:dind
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  script:
    - docker build --pull -t "$CI_REGISTRY_IMAGE" .
    - docker push "$CI_REGISTRY_IMAGE"
  only:
    - master
```

Figure 41: `.gitlab-ci.yml` file for the crop_image microservice

8.4 Airflow

The execution of our pipeline is controlled by Apache Airflow. Given Airflow's structure, we created the following workflow artifacts tables:

service id	service name	service description	image url	cluster name	owned by
1	validate_input	Validates and populates the metadata into Postgresql	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_validate	CS 5604 ETD 2020 – testing	ETD
2	pdf_to_image	Converts each page in a PDF to an image	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_pdf_to_image	CS 5604 ETD 2020 – testing	ETD
3	image_inference	Creates bounding boxes for each page	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_image_inference	CS 5604 ETD 2020 – testing	ETD
4	crop_image	Crops each figure using the bounding boxes	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_crop_image	CS 5604 ETD 2020 – testing	ETD
5	chapter_extraction	Segments the PDF into its appropriate chapters as PDF	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_chapter_extraction	CS 5604 ETD 2020 – testing	ETD
6	etd_text_extraction	Extracts text from both the ETD PDF and the Chapter PDF files	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_text_extraction	CS 5604 ETD 2020 – testing	ETD
7	etd_probabilistic _classification	Generates subject labels for each ETD & chapter text	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_probabalistic _classification	CS 5604 ETD 2020 – testing	ETD
8	etd_metadata _ingestion	Ingests and indexes metadata from Postgres into Elasticsearch for frontend consumption	container.cs.vt.edu/cs-5604-fall-2020/etd/etd_metadata_ingestion	CS 5604 ETD 2020 – testing	ETD

Table 7: Airflow service descriptions

Initial Input (Source)	goal_id	goal_name	goal_description	goal_format	file_location	environment_variable	owned_by
User	1	etd_pdf	etd pdf	<PDF file>	/mnt/nfs/etd/ etd.pdf	ETD_PDF	ETD
User	2	handle_id	handle number	<JSON file>	/mnt/nfs/etd/ handle.json	HANDLE_ID	ETD
User	3	document_type	dissertation or thesis or empty	<JSON file>	/mnt/nfs/etd/ doc_type.json	DOC_TYPE	ETD
User	4	etd_metadata	metadata for etd	<TXT file>	/mnt/nfs/etd/ etd_metadata.txt	ETD_METADATA	ETD
N/A	5	validate_finish	file to indicate input validation has been completed	<JSON file>	/mnt/nfs/etd/ validate_finish.json	VALIDATE_FINISH	ETD
N/A	6	images_of_pages	directory to store all pages of PDF as images	<Directory>	/mnt/nfs/etd/ images_of_pages	IMAGES_OF_PAGES	ETD
N/A	7	pages_with_detected_boxes	directory to store bounding boxes for each image within each page of the pdf	<Directory>	/mnt/nfs/etd/ bounding_boxes	BOUNDING_BOXES	ETD
N/A	8	tables_and_figures	directory to store final tables and figures as images	<Directory>	/mnt/nfs/etd/ tab_fig	TABLES_FIGURES	ETD
N/A	9	chapters_pdf	directory to store all extracted chapter PDF files	<Directory>	/mnt/nfs/etd/ chapters_pdf	CHAPTERS_PDF	ETD
N/A	10	etd_text	full text extracted from etd PDF	<TXT file>	/mnt/nfs/etd/etd.txt	ETD_TEXT	ETD
User	11	etd_dublin_core	dublin core formatted metadata for etds	<XML file>	/mnt/nfs/etd/ dublin_core.xml	DUBLIN_CORE	ETD
N/A	12	chapters_text	directory to store full text for each extracted chapter	<Directory>	/mnt/nfs/etd/ chapters_text	CHAPTERS_TEXT	ETD
N/A	13	classify_finish	file to indicate classification has been completed	<JSON file>	/mnt/nfs/etd/ classify_finish.json	CLASSIFY_FINISH	ETD
N/A	14	pipeline_finish	file to indicate ingestion has been completed	<JSON file>	/mnt/nfs/etd/ pipeline_finish.json	PIPELINE_FINISH	ETD
INT Airflow	0	workflow_id	airflow run id	N/A	N/A	WORKFLOW_ID	INT
Docker	0	postgres_conn_string	database connection string	N/A	N/A	POSTGRES_CONN_STRING	ETD

Table 8: Airflow goals

goal_id	service_id	input_goal_id
5	1	1
5	1	2
5	1	3
5	1	4
6	2	1
6	2	2
6	2	3
7	3	6
7	3	2
7	3	3
8	4	7
8	4	2
8	4	3
9	5	1
9	5	2
9	5	3
10	6	1
10	6	2
10	6	3
10	6	9
12	6	1
12	6	2
12	6	3
12	6	9
13	7	2
13	7	3
13	7	11
13	7	10
13	7	12
14	8	2
14	8	3
14	8	13

Table 9: Airflow reasoner table

Services Table Table 7 displays all of the services offered by the ETD team and the associated ID number as stored by Airflow.

Goals Table Table 8 displays all of the input and output data points (displayed as goals) consumed and generated by the services. The data inputs passed in by the user or other teams are indicated by the Initial Input (Source) column. Samples of each goal are listed in Appendix A.

Reasoner Table Table 9 connects the input/output goals to the services. For example: service with ID 1, receives goals with IDs 1, 2, 3 and 4 as input and returns goal ID 5 as output. That is, the validate service takes in the ETD PDF file, handle_id, document type, and existing ETD Metadata, updates Postgres database and outputs a file indicating that the task has been finished.

Using the Reasoner table, Airflow knows what files to input to the services and what files to expect when a service is completed in order to proceed in the pipeline. For example, once Airflow notices the validate_finish file has been created, it starts the service dependent on validate_finish which includes the pdf_to_image service and chapter_extraction service.

8.5 Adding Microservices

By using Airflow, future researchers have the ability to add microservices to our existing pipeline. This section is a overview of the steps needed to add a microservice to our pipeline.

1. Create a new repository for the microservice and implement it's functionality. Input and output should be controlled through environment variables.

2. Create unit tests for the microservice.

3. Create a `.gitlab-ci.yml` file. This file will define the Gitlab Runner configuration. The `.gitlab-ci.yml` files in other microservice repositories can be referenced as examples. Of note are the build stages of the configuration. Like in Figure 41, the Gitlab Runner has the ability to containerize, successful build, and add the to the repository's registry automatically.

4. After defining the Gitlab Runner configuration, push the branch to start the test pipeline. Any branch with a .gitlab-ci.yml file in it with run a test pipeline when commits are pushed. Pipelines can seen from the Gitlab website.

5. Create a deploy token for the service. This token will be used by Airflow or other workflow management software to pull the images from the repository. If you know you are going to use the token for multiple things, make sure to write down the username and password.

6. Add the image to the team's namespace on `rancher.cs.vt.edu`. Deploying the containers here makes them usable for Airflow. After deployment, the number of pods can be set to zero. This prevents the containers from getting stuck in a restart loop. The Fall 2020 INT Team report has more detailed instructions on deploying containers and using Rancher.

7. Once the image is on Airflow, the Airflow reasoner needs to be updated with the image's location, the new goals; some existing goals may also have to be modified to account for the new microservice.

8. Since the FE team is responsible for what the end-user sees, they might have to be contacted in case the changes to the workflow do not appear on the website.

References

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[2] Charu C Aggarwal. *Machine learning for text*. Springer, 2018.

[3] Naman Ahuja, Ritesh Bansal, William A. Ingram, Palakh Jude, Sampanna Kahu, and Xinyue Wang. Big Data Text Summarization: Using Deep Learning to Summarize Theses and Dissertations. CS4984: Special Topics Fall 2018 Team 16, Virginia Tech, Blacksburg, VA, 2018 http://hdl.handle.net/10919/86406.

[4] Hanan M Alghamdi, Ali Selamat, and Nor Shahriza Abdul Karim. Arabic web pages clustering and annotation using semantic class features. *Journal of King Saud University-Computer and Information Sciences*, 26(4):388–397, 2014.

[5] Isaac Alpizar-Chacon and Sergey Sosnovsky. Order out of chaos: Construction of knowledge models from PDF textbooks. In *Proceedings of the ACM Symposium on Document Engineering 2020*, pages 1–10, 2020.

[6] John Aromando, Bipasha Banerjee, William A. Ingram, Palakh Jude, and Sampanna Kahu. Classification and extraction of information from ETD documents. CS6604: Digital Libraries Fall 2019, Virginia Tech, Blacksburg, VA, 2020. http://hdl.handle.net/10919/96645.

[7] Horst Bunke and Kaspar Riesen. Recent advances in graph-based pattern recognition with applications in document analysis. *Pattern Recognition*, 44(5):1057–1067, 2011.

[8] Christopher Clark. Pdffigures2.0. GitHub, Seattle, WA, 2019. https://github.com/allenai/pdffigures2 (accessed Oct. 7, 2020).

[9] Sneha S Desai and JA Laxminarayana. WordNet and semantic similarity based approach for document clustering. In *2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, pages 312–317. IEEE, 2016.

[10] Elastic. Elasticsearch. Elasticsearch B.V., Mountain View, CA, 2020. https://elastic.co (accessed Oct. 7, 2020).

[11] Facebook. fastText. Facebook Inc., Menlo Park, CA, 2020. https://fastText.cc/ (accessed Oct. 20, 2020).

[12] Nicolas Fiorini, Sébastien Harispe, Sylvie Ranwez, Jacky Montmain, and Vincent Ranwez. Fast and reliable inference of semantic clusters. *Knowledge-Based Systems*, 111:133–143, 2016.

[13] Chris Fournier. Evaluating text segmentation using boundary edit distance. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1702–1712, 2013.

[14] Olli-Pekka Heinisuo. opencv-python. GitHub, Tampere, Finland, 2020. https://github.com/skvark/opencv-python (accessed Dec. 01, 2020).

[15] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Palakh Mignonne Jude. Increasing Accessibility of Electronic Theses and Dissertations (ETDs) Through Chapter-level Classification. Master's thesis, Virginia Tech, Blacksburg, VA, 2020. http://hdl.handle.net/10919/99294.

[18] Sampanna Kahu. deepfigures-open. GitHub, Blacksburg, VA, 2020. https://github.com/SampannaKahu/deepfigures-open (accessed Oct. 20, 2020).

[19] Sampanna Yashwant Kahu. Figure Extraction from Scanned Electronic Theses and Dissertations. Master's thesis, Virginia Tech, Blacksburg, VA, 2020. http://hdl.handle.net/10919/100113.

[20] Kulendra Kuma Kaushal, Rutwik Kulkarni, Aarohi Sumant, Chaoran Wang, Chenhan Yuan, and Liling Yuan. Collection Management of Electronic Theses and Dissertations (CME) CS5604 fall 2019. CS5604: Information Retrieval Fall 2019, Virginia Tech, Blacksburg, VA, 2019. http://hdl.handle.net/10919/96484.

[21] Wahab Khan, Ali Daud, Khairullah Khan, Jamal Abdul Nasir, Mohammed Basheri, Naif Aljohani, and Fahd Saleh Alotaibi. Part of speech tagging in Urdu: Comparison of machine and deep learning approaches. *IEEE Access*, 7:38918–38936, 2019.

[22] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[23] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. Cornell University, 2014.

[24] Yuan Li, Satvik Chekuri, Tianrui Hu, Soumya Arvind Kumar, and Nicholas Gill. Elasticsearch (ELS) CS5604 Fall 2019. CS5604: Information Retrieval Fall 2019, Virginia Tech, Blacksburg, VA, 2019. http://hdl.handle.net/10919/96310.

[25] Yung-Shen Lin, Jung-Yi Jiang, and Shie-Jue Lee. A similarity measure for text classification and clustering. *IEEE transactions on knowledge and data engineering*, 26(7):1575–1590, 2013.

[26] Patrice Lopez. GROBID. GitHub, 2008-2020. https://github.com/kermitt2/grobid (accessed Oct. 7, 2020).

[27] Christopher D Manning, Hinrich Schütze, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge University Press, 2008.

[28] Rifat Sabbir Mansur, Prathamesh Mandke, Jiaying Gong, Sandhya M Bharadwaj, Adheesh Sunil Juvekar, and Sharvari Chougule. Text Analytics and Machine Learning (TML) CS5604 Fall 2019. CS5604: Information Retrieval Fall 2019, Virginia Tech, Blacksburg, VA, 2019. http://hdl.handle.net/10919/96226.

[29] Jorj McKie and Ruikai Liu. PyMuPDF. GitHub, 2020. https://github.com/pymupdf/PyMuPDF (accessed Oct. 7, 2020).

[30] Gupta Mehul, Patel Ankita, Dave Namrata, Goradia Rahul, and Saurin Sheth. Text-based image segmentation methodology. *Procedia Technology*, 14:465–472, 2014.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[32] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[33] Lev Pevzner and Marti A Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36, 2002.

[34] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333, 2011.

[35] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[36] Jakub Roztocil. HTTPie. HTTPie, 2012-2020. https://httpie.io/ (accessed Oct. 20, 2020).

[37] Noah Siegel, Nicholas Lourie, Russell Power, and Waleed Ammar. deepfigures-open. GitHub, 2018. https://github.com/allenai/deepfigures-open (accessed Oct. 7, 2020).

[38] Matthew Stamy. PyPDF2. GitHub, Plano, TX, 2018. https://github.com/mstamy2/PyPDF2 (accessed Oct. 7, 2020).

[39] Mark Steyvers and Tom Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

[40] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[41] Gianmaria Tarantino, Stefania Monica, and Federico Bergenti. A probabilistic matrix factorization algorithm for approximation of sparse matrices in natural language processing. *ICT Express*, 4(2):87–90, 2018.

[42] Lena Tenenboim-Chekina, Lior Rokach, and Bracha Shapira. Identification of Label Dependencies for Multi-label Classification. *Working Notes*, page 53, 2010.

[43] VTechWorks. ETDs: Virginia Tech Electronic Theses and Dissertations. Virginia Tech, Blacksburg, VA, 2020. http://vtechworks.lib.vt.edu/handle/10919/5534 (accessed Oct. 5, 2020).

[44] Tingting Wei, Yonghe Lu, Huiyou Chang, Qiang Zhou, and Xianyu Bao. A semantic approach for text clustering using WordNet and lexical chains. *Expert Systems with Applications*, 42(4):2264–2275, 2015.

[45] Ruiguo Yu, Jie Gao, Mei Yu, Wenhuan Lu, Tianyi Xu, Mankun Zhao, Jie Zhang, Ruixuan Zhang, and Zhuo Zhang. LSTM-EFG for wind power forecasting based on sequential correlation features. *Future Generation Computer Systems*, 93:33–42, 2019.

[46] Min-Ling Zhang and Zhi-Hua Zhou. ML-kNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038 – 2048, 2007.

A Appendix

A.1 Elasticsearch metadata mapping

This section is a listing of the JSON used to create the ETD metadata index.

```
{
    "mappings": {
        "properties": {
            "chapters": {
                "properties": {
                    "uid" : {"type": "integer"},
                    "subject": {"type": "keyword"},
                    "pages": {"type": "integer"},
                    "uri": {"type": "keyword"}
                }
            },
            "contributor_author": {
                "type": "keyword",
                "index": "true"
            },
            "contributor_committeechair": {
                "type": "keyword",
                "index": "true"
            },
            "contributor_committeecochair": {
                "type": "keyword",
                "index": "true"
            },
            "contributor_committeemember": {
                "type": "keyword",
                "index": "true"
            },
            "contributor_department": {
                "type": "keyword",
                "index": "true"
            },
            "date_accessioned": {
                "type": "date",
                "index": "false"
            },
            "date_available": {
                "type": "date",
                "index": "false"
            },
            "date_issued": {
```

```
        "type": "date",
        "index": "true"
    },
    "date_adate": {
        "type": "date",
        "index": "false"
    },
    "date_rdate": {
        "type": "date",
        "index": "false"
    },
    "date_sdate": {
        "type": "date",
        "index": "false"
    },
    "identifier_other": {
        "type": "text",
        "index": "true"
    },
    "identifier_uri": {
        "type": "keyword",
        "index": "false"
    },
    "description_abstract": {
        "type": "text",
        "index": "true"
    },
    "description_provenance": {
        "type": "text",
        "index": "false"
    },
    "format_medium": {
        "type": "text"
    },
    "publisher": {
        "type": "text",
        "index": "true"
    },
    "rights": {
        "type": "text",
        "index": "false"
    },
    "subject": {
        "type": "text",
        "index": "true"
```

```
},
"subject_lcc": {
    "type": "text",
    "index": "true"
},
"subject_lcsh": {
    "type": "text",
    "index": "true"
},
"title": {
    "type": "text",
    "index": "true"
},
"type": {
    "type": "keyword",
    "index": "true"
},
"description_degree": {
    "type": "keyword",
    "index": "true"
},
"degree_name": {
    "type": "keyword",
    "index": "true"
},
"degree_level": {
    "type": "keyword",
    "index": "true"
},
"degree_grantor": {
    "type": "text",
    "index": "true"
},
"degree_disclipline": {
    "type": "keyword",
    "index": "true"
},
"description_sponsorship": {
    "type": "text",
    "index": "false"
},
"handle": {
    "type": "keyword",
    "index": "true"
},
```

```
            "relation_haspart": {
                "type": "text",
                "index": "false"
            },
            "relation": {
                "type": "text",
                "index": "false"
            },
            "identifier_sourceurl": {
                "type": "text",
                "index": "false"
            },
            "identifier_oclc": {
                "type": "text",
                "index": "true"
            },
            "etd_filename": {
                "type": "text",
                "index": "false"
            },
            "etd_uri": {
                "type": "keyword",
                "index": "true"
            },
            "figures_folder_uri": {
                "type": "keyword",
                "index": "true"
            },
            "figures_total_count": {
                "type": "integer",
                "index": "false"
            },
            "tables_folder_uri": {
                "type": "keyword",
                "index": "true"
            },
            "tables_total_count": {
                "type": "integer",
                "index": "false"
            }
        }
    }
}
```

A.2 Sample goals

Full Scale Experimental Transonic Fan Interaction with a Boundary Layer Ingesting Total Pressure Distortion

Justin Mark Bailey

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Mechanical Engineering

Walter F. O'Brien, Chair
Clinton L. Dancey
Wing F. Ng
Kevin T. Lowe
Alfred L. Wicks

November 7, 2016
Blacksburg, Virginia

Keywords: Experimental Engine Testing, Distortion, Interaction, Total Pressure, Boundary Layer Ingesting

Figure 42: Sample Goal ID 1 - `etd.pdf`

```
{"handle": 73987}
```

Figure 43: Sample Goal ID 2 - Handle ID of dissertation

```
{"doc_type": "D"}
```

Figure 44: Sample Goal ID 3 - Document Type ("D" for "Dissertation")

```
contributor-author: Bailey, Justin Mark
date-accessioned: 2017-01-06T13:34:06Z
date-available: 2017-01-06T13:34:06Z
date-issued: 2017-01-05
identifier-other: vt_gsexam:9274
identifier-uri: http://hdl.handle.net/10919/73987
description-abstract: Future commercial transport aircraft will feature more aerodynamic architectures to
    accommodate stringent design goals for higher fuel efficiency, reduced cruise and taxi NOx emissions, and
    reduced noise. Airframe designs most likely to satisfy the first goal feature architectures that lead to
    the formation of non-uniform flow introduced to the engine through boundary layer ingesting (BLI) inlets,
    creating a different operational environment from which the engines were originally designed. The goal of
    this study was to explore the effects such non-uniform flow would have on the behavior and performance of a
    transonic fan in a full scale engine test environment. This dissertation presents an experimental study of
    the interaction between a full scale transonic fan and a total pressure distortion representative of a
    boundary layer ingesting serpentine inlet. A five-hole pneumatic probe was traversed directly in front of
    and behind a fan rotor to fully characterize the inlet and outlet fan profile. The distortion profile was
    also measured at the aerodynamic interface plane (AIP) with an SAE standard total pressure rake, which has
    historically been accepted as the inlet profile to the fan. This provided a comparison between the present
    work and current practice. Accurate calculation of local fan performance metrics such as blade loading,
    pressure rise, and efficiency were obtained. The fan inlet measurement profile greatly enhanced the
    understanding of the fan interaction to the flow distortion and provided a more complete explanation of the
    fan behavior. Secondary flowfield formation due to the accelerated flow redistribution directly upstream of
    the fan created localized bulk co- and counter- rotating swirl regions that were found to be correlated
    with localized fan performance phenomena. It was observed that the effects of the distortion on fan
    performance were exaggerated if the assumed fan inlet profiles were data taken only at the AIP. The
    reduction in fan performance with respect to undistorted inlet conditions is also explored, providing
    insight into how such distortions can be compared to baseline conditions. The dissertation closes with
    several recommendations for improving distortion tolerant fan design in the context of experimental
    research and development.
description-provenance: Author Email: jmb@vt.edu
description-provenance: Made available in DSpace on 2017-01-06T13:34:06Z (GMT). No. of bitstreams: 1
    Bailey_JM_D_2017.pdf: 9128042 bytes, checksum: 7438e886322739e17247ed2c907658b0 (MD5)   Previous issue date
    : 2017-01-05
format-medium: ETD
publisher-none: Virginia Tech
```

Figure 45: Sample Goal ID 4 - ETD Metadata

```
{"validate_finish": True}
```

Figure 46: Sample Goal ID 5 - "Validate Finish" JSON created once the validation service has completed

Chapter 2 - Experimental Methodology

rotation, placing significant additional wear on the fan blade structure. This effect is further exacerbated for multiple-per-rev distortions.

2.4.1.1 Single-per-rev Excitation

The distortion pattern under study for this work was an intense, single-per-rev distortion, thus caution must be exercised to avoid exciting blade resonating frequencies. Shown in Figure 2.14 are the results of a modal analysis performed on the blade geometry to obtain the excitation frequencies, which is detailed in Appendix D. These frequencies were then compared to the engine order excitations via a Campbell diagram. Only the second engine order poses a serious concern as it intersects with the 1st mode frequency of the blade at approximately 61% fan speed, indicated in Figure 2.14-left. Care was taken during testing to ensure the engine avoided extensive operation at this condition, which was accomplished by reserving testing to cooler days when the uncorrected fan speed would be lower than 60% to match corrected conditions.

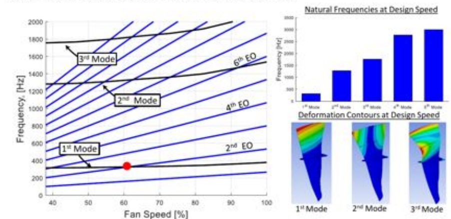


Figure 2.14. Campbell diagram (left) with accompanying modal frequency results at design speed (top-right) and deformation (bottom-right).

2.4.1.2 Metal Angles

To gain an understanding of the undistorted operating conditions of the engine it was necessary to obtain the metal angles of the fan blade. This was done by performing a three dimensional scan of the blade, breaking up the span into discrete sections, and then constructing the camber line at each section to obtain the leading and trailing edge angles. Figure 2.15 summarizes the process and results of this effort. A more detailed explanation of the process used, as well as a data table of the reversed engineered blade geometry, can be found in Appendix E.

17

Appendix G - Multiple Speed Data

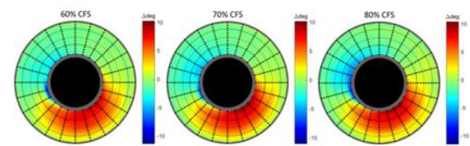


Figure G.6. Station 4 incidence angle difference from undistorted for all fan speeds investigated.

G.3 Station 5

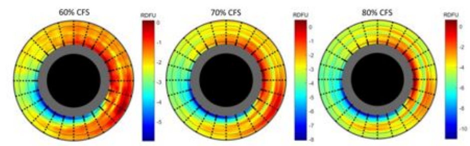


Figure G.7. Relative Station 5 total pressure from undistorted for all fan speeds investigated.

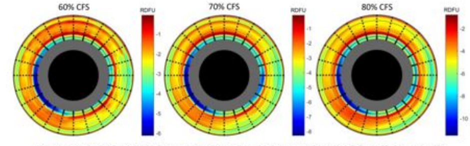
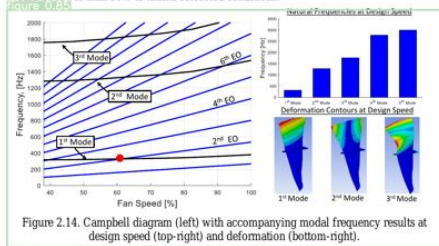
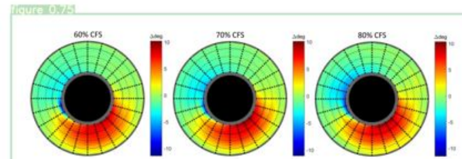


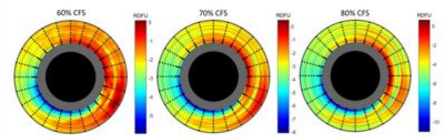
Figure G.8. Relative Station 5 static pressure from undistorted for all fan speeds investigated.

89

Figure 47: Sample Goal ID 6 - Directory of Images for each page in PDF







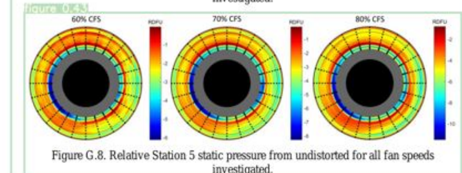


Figure 48: Sample Goal ID 7 - (a) Bounding boxes overlaid for each Image in Goal ID 6 and (b) Bounding box value in text files

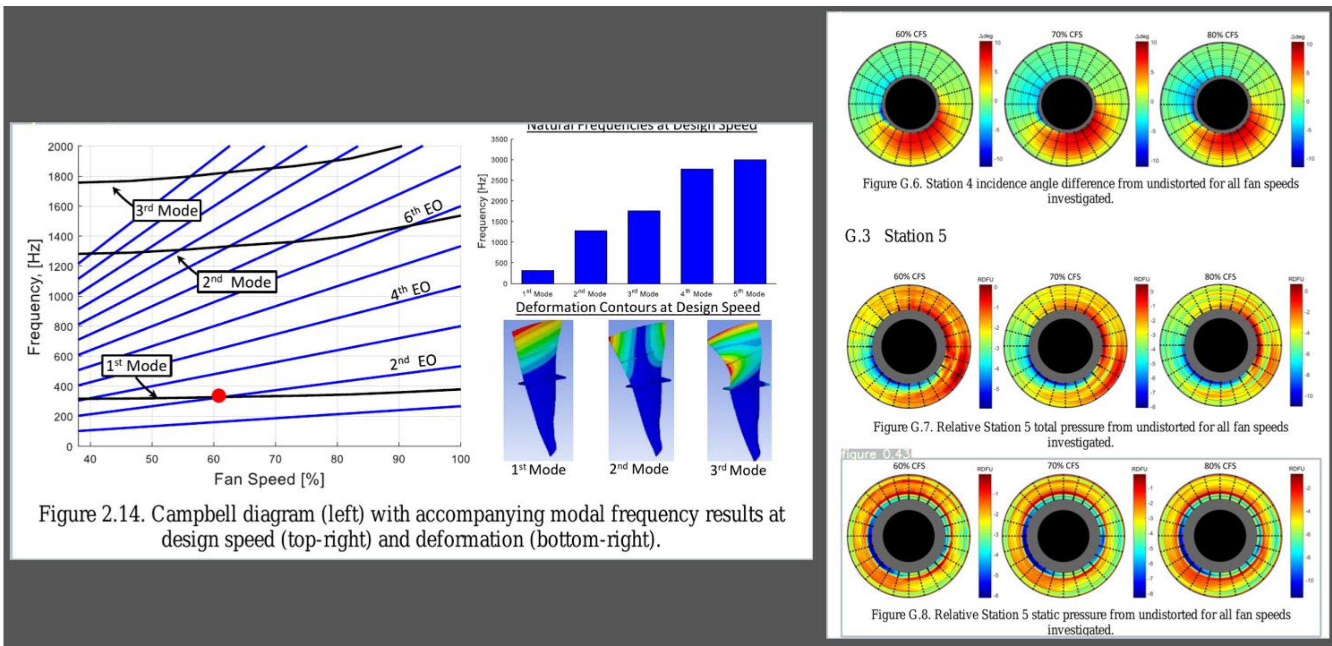


Figure 49: Sample Goal ID 8 - Pictures extracted from the bounding boxes in Goal ID 7

Chapter 1 - Introduction

Chapter 1 Introduction

In order to meet stringent requirements for next generation commercial aviation, aircraft will need to employ more aerodynamic designs to 1) reduce noise, 2) improve landing, takeoff, and cruise emissions, and 3) improve fuel consumption [1,2]. These designs are expected to take the aviation industry away from traditional 'tube-and-wing' architectures into a more hybrid or blended wing body (HWB, BWB) design that will satisfy aircraft operation improvement goals. A comparison between modern and proposed airframe architectures is shown in Figure 1.1.

Figure 1.1. Modern aircraft 'tube-and-wing' design (Left) and future blended wing design (Right).

A common feature of these HWB airframes is the integration of the engines with the aircraft fuselage. This relocation changes the inlet flow environment as the engines are no longer mounted below the wings where uniform inlet flow is ingested for the majority of the flight envelope. Mounting the engines in close proximity to the airframe, or embedding them within the airframe, results in non-uniform (distorted) inlet flow caused by the airframe body or inlet ducts. In the interest of performance and efficiency, modern commercial transport engines are designed to tolerate only small levels of distortion, such as during takeoff or strong crosswind conditions. Integrated propulsion systems will experience distorted inflow over the entire flight envelope and their interaction with such conditions will need to be studied to determine how engine performance is affected, and to support the design of distortion-tolerant engines.

To determine how well engines can withstand such an environment, three criteria will need to be evaluated in great detail; aeromechanics, stability, and performance. Although studies have shown that engine operation penalties are found in each one of these, the overall benefit of highly integrated airframe and propulsion systems can justify the operational challenges introduced to the engines [4].

1

Chapter 2 - Experimental Methodology

Chapter 2 Experimental Methodology

Obtaining accurate measurements on a full scale turbofan engine requires many custom, tailor-fit components to be designed, built, calibrated, and installed. A basic overview of the equipment and measurement protocol utilized will be discussed here. Interested readers are referred to related work that examines these components in further detail [20].

2.1 Experimental Investigation

Full-scale experimental total pressure distortion ground tests were conducted on a modified Pratt & Whitney Canada JT15D-1 turbofan engine at the Virginia Tech Turbomachinery and Propulsion Laboratory in Blacksburg, Virginia. A schematic of the experimental setup is shown in Figure 2.1.

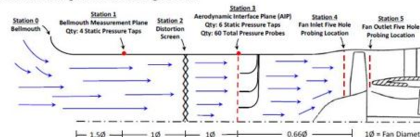


Figure 2.1. Schematic of experimental components and configuration.

The areas of interest are divided into six stations:

- Station 0 - Calibrated bellmouth inlet that funnels airflow from outside the test cell and into the experimental rig.
- Station 1 - Bellmouth measurement plane with four static pressure taps equally spaced circumferentially around the tunnel.
- Station 2 - Distortion screen that produces a total pressure distortion representative of a boundary layer ingesting (BLI) serpentine inlet.
- Station 3 - AIP rake case with 60 local total pressure probes and six equally spaced circumferential wall static pressure taps.
- Station 4 - Fan inlet measurement plane located approximately 1 inch upstream of the fan leading edge (~40% chord).
- Station 5 - Fan outlet measurement plane located approximately 0.375" downstream of the fan trailing edge (~23% chord).

5

Figure 50: Sample Goal ID 9 - 2 of 28 Chapter PDF files generated for Sample Goal ID 1

Bailey_JM_D_2017 - Notepad

File Edit Format View Help

Full Scale Experimental Transonic Fan Interaction with a
Boundary Layer Ingesting Total Pressure Distortion

Justin Mark Bailey

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
In
Mechanical Engineering

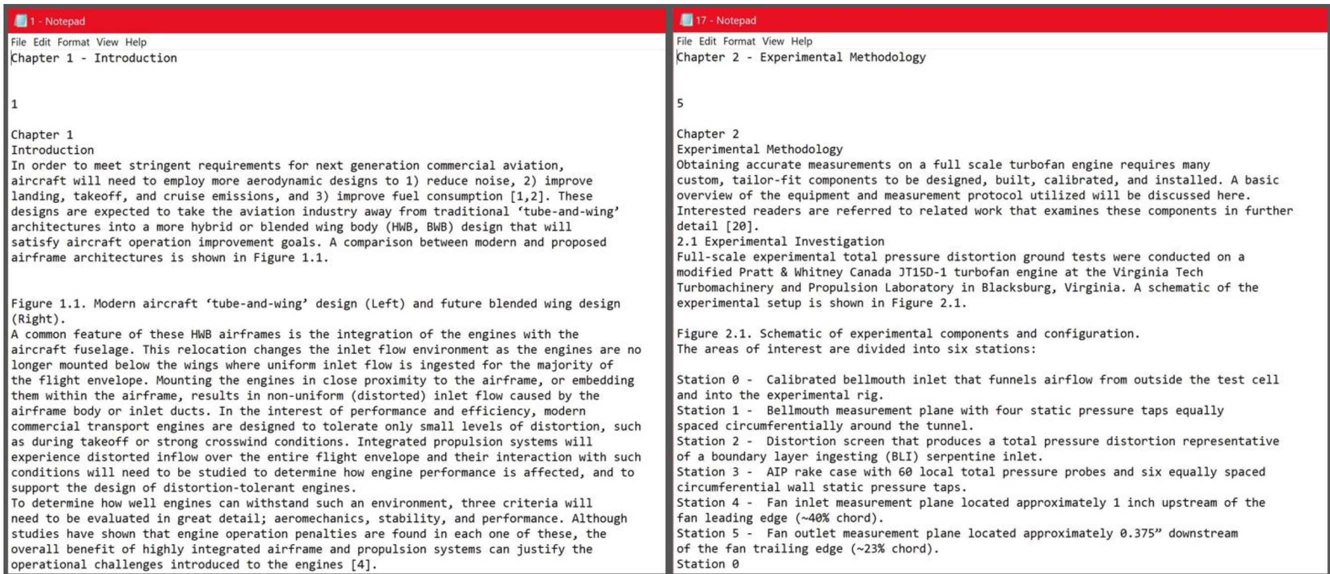
Walter F. O’Brien, Chair
Clinton L. Dancey
Wing F. Ng
Kevin T. Lowe
Alfred L. Wicks

November 7, 2016
Blacksburg, Virginia

Figure 51: Sample Goal ID 10 - Text extracted from Sample Goal ID 1

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<dublin_core schema="dc">
  <dcvalue element="contributor" qualifier="author" language="en_US">Bailey,&#x20;Justin&#x20;Mark</dcvalue>
  <dcvalue element="date" qualifier="accessioned">2017-01-06T13:34:06Z</dcvalue>
  <dcvalue element="date" qualifier="available">2017-01-06T13:34:06Z</dcvalue>
  <dcvalue element="date" qualifier="issued" language="en_US">2017-01-05</dcvalue>
  <dcvalue element="identifier" qualifier="other" language="en_US">vt_gsexam:9274</dcvalue>
  <dcvalue element="identifier" qualifier="uri">http:&#x2F;&#x2F;hdl.handle.net&#x2F;10919&#x2F;73987</dcvalue>
  <dcvalue element="description" qualifier="abstract" language="en_US">Future&#x20;commercial&#x20;transport&#x20;aircraf
20;fuel&#x20;efficiency,&#x20;reduced&#x20;cruise&#x20;and&#x20;taxi&#x20;NOx&#x20;emissions,&#x20;and&#x20;reduced&#x20;
d&#x20;to&#x20;the&#x20;formation&#x20;of&#x20;non-uniform&#x20;flow&#x20;introduced&#x20;to&#x20;the&#x20;engine&#x20;th
ich&#x20;the&#x20;engines&#x20;were&#x20;originally&#x20;designed.&#x20;The&#x20;goal&#x20;of&#x20;this&#x20;study&#x20;wa
nce&#x20;of&#x20;a&#x20;transonic&#x20;fan&#x20;in&#x20;a&#x20;full&#x20;scale&#x20;engine&#x20;test&#x20;environment.&#x
transonic&#x20;fan&#x20;and&#x20;a&#x20;total&#x20;pressure&#x20;distortion&#x20;representative&#x20;of&#x20;a&#x20;bounda
front&#x20;of&#x20;and&#x20;behind&#x20;a&#x20;fan&#x20;rotor&#x20;to&#x20;fully&#x20;characterize&#x20;the&#x20;inlet&#x2
face&#x20;plane&#x20;(AIP)&#x20;with&#x20;an&#x20;SAE&#x20;standard&#x20;total&#x20;pressure&#x20;rake,&#x20;which&#x20;ha
on&#x20;between&#x20;the&#x20;present&#x20;work&#x20;and&#x20;current&#x20;practice.&#x0A;Accurate&#x20;calculation&#x20;
x20;obtained.&#x20;The&#x20;fan&#x20;inlet&#x20;measurement&#x20;profile&#x20;greatly&#x20;enhanced&#x20;the&#x20;understa
nation&#x20;of&#x20;the&#x20;fan&#x20;behavior.&#x20;Secondary&#x20;flowfield&#x20;formation&#x20;due&#x20;to&#x20;the&#x2
20;counter-&#x20;rotating&#x20;swirl&#x20;regions&#x20;that&#x20;were&#x20;found&#x20;to&#x20;be&#x20;correlated&#x20;with
20;on&#x20;fan&#x20;performance&#x20;were&#x20;exaggerated&#x20;if&#x20;the&#x20;assumed&#x20;fan&#x20;inlet&#x20;profiles
o&#x20;undistorted&#x20;inlet&#x20;conditions&#x20;is&#x20;also&#x20;explored,&#x20;providing&#x20;insight&#x20;into&#x20
everal&#x20;recommendations&#x20;for&#x20;improving&#x20;distortion&#x20;tolerant&#x20;fan&#x20;design&#x20;in&#x20;the&#x
  <dcvalue element="description" qualifier="provenance" language="en_US">Author&#x20;Email:&#x20;jmb@vt.edu</dcvalue>
  <dcvalue element="description" qualifier="provenance" language="en">Made&#x20;available&#x20;in&#x20;DSpace&#x20;on&#x20
22739e17247ed2c907658b0&#x20;(MD5)&#x0A;&#x20;&#x20;Previous&#x20;issue&#x20;date:&#x20;2017-01-05</dcvalue>
  <dcvalue element="format" qualifier="medium" language="en_US">ETD</dcvalue>
  <dcvalue element="publisher" qualifier="none" language="en_US">Virginia&#x20;Tech</dcvalue>
  <dcvalue element="rights" qualifier="none" language="en_US">This&#x20;item&#x20;is&#x20;protected&#x20;by&#x20;copyrigh
x20;by&#x20;law&#x20;even&#x20;without&#x20;permission&#x20;from&#x20;the&#x20;rights&#x20;holder(s),&#x20;or&#x20;the&#x2
x20;uses&#x20;you&#x20;need&#x20;to&#x20;obtain&#x20;permission&#x20;from&#x20;the&#x20;rights&#x20;holder(s).</dcvalue>
  <dcvalue element="subject" qualifier="none" language="en_US">Experimental&#x20;Engine&#x20;Testing</dcvalue>
  <dcvalue element="subject" qualifier="none" language="en_US">Distortion</dcvalue>
  <dcvalue element="subject" qualifier="none" language="en_US">Interaction</dcvalue>
  <dcvalue element="subject" qualifier="none" language="en_US">Total&#x20;Pressure</dcvalue>
  <dcvalue element="subject" qualifier="none" language="en_US">Boundary&#x20;Layer&#x20;Ingesting</dcvalue>
  <dcvalue element="title" qualifier="none" language="en_US">Full&#x20;Scale&#x20;Experimental&#x20;Transonic&#x20;Fan&#x2
  <dcvalue element="type" qualifier="none" language="en_US">Dissertation</dcvalue>
  <dcvalue element="contributor" qualifier="department" language="en_US">Mechanical&#x20;Engineering</dcvalue>
  <dcvalue element="description" qualifier="degree" language="en_US">PHD</dcvalue>
  <dcvalue element="contributor" qualifier="committeechair" language="en_US">O&#39;Brien,&#x20;Walter&#x20;F</dcvalue>
  <dcvalue element="contributor" qualifier="committeemember" language="en_US">Dancey,&#x20;Clinton&#x20;L</dcvalue>
  <dcvalue element="contributor" qualifier="committeemember" language="en_US">Lowe,&#x20;Kevin&#x20;T</dcvalue>
  <dcvalue element="contributor" qualifier="committeemember" language="en_US">Wicks,&#x20;Alfred&#x20;L</dcvalue>
  <dcvalue element="contributor" qualifier="committeemember" language="en_US">Ng,&#x20;Wing&#x20;Fai</dcvalue>
</dublin_core>
```


Figure 52: Sample Goal ID 11 - Dublin Core XML



1 - Notepad
File Edit Format View Help

Chapter 1 - Introduction

1

Chapter 1
Introduction
In order to meet stringent requirements for next generation commercial aviation,
aircraft will need to employ more aerodynamic designs to 1) reduce noise, 2) improve
landing, takeoff, and cruise emissions, and 3) improve fuel consumption [1,2]. These
designs are expected to take the aviation industry away from traditional 'tube-and-wing'
architectures into a more hybrid or blended wing body (HWB, BWB) design that will
satisfy aircraft operation improvement goals. A comparison between modern and proposed
airframe architectures is shown in Figure 1.1.

Figure 1.1. Modern aircraft 'tube-and-wing' design (Left) and future blended wing design
(Right).
A common feature of these HWB airframes is the integration of the engines with the
aircraft fuselage. This relocation changes the inlet flow environment as the engines are no
longer mounted below the wings where uniform inlet flow is ingested for the majority of
the flight envelope. Mounting the engines in close proximity to the airframe, or embedding
them within the airframe, results in non-uniform (distorted) inlet flow caused by the
airframe body or inlet ducts. In the interest of performance and efficiency, modern
commercial transport engines are designed to tolerate only small levels of distortion, such
as during takeoff or strong crosswind conditions. Integrated propulsion systems will
experience distorted inflow over the entire flight envelope and their interaction with such
conditions will need to be studied to determine how engine performance is affected, and to
support the design of distortion-tolerant engines.
To determine how well engines can withstand such an environment, three criteria will
need to be evaluated in great detail; aeromechanics, stability, and performance. Although
studies have shown that engine operation penalties are found in each one of these, the
overall benefit of highly integrated airframe and propulsion systems can justify the
operational challenges introduced to the engines [4].

17 - Notepad
File Edit Format View Help

Chapter 2 - Experimental Methodology

5

Chapter 2
Experimental Methodology
Obtaining accurate measurements on a full scale turbofan engine requires many
custom, tailor-fit components to be designed, built, calibrated, and installed. A basic
overview of the equipment and measurement protocol utilized will be discussed here.
Interested readers are referred to related work that examines these components in further
detail [20].
2.1 Experimental Investigation
Full-scale experimental total pressure distortion ground tests were conducted on a
modified Pratt & Whitney Canada JT15D-1 turbofan engine at the Virginia Tech
Turbomachinery and Propulsion Laboratory in Blacksburg, Virginia. A schematic of the
experimental setup is shown in Figure 2.1.

Figure 2.1. Schematic of experimental components and configuration.
The areas of interest are divided into six stations:

Station 0 - Calibrated bellmouth inlet that funnels airflow from outside the test cell
and into the experimental rig.
Station 1 - Bellmouth measurement plane with four static pressure taps equally
spaced circumferentially around the tunnel.
Station 2 - Distortion screen that produces a total pressure distortion representative
of a boundary layer ingesting (BLI) serpentine inlet.
Station 3 - AIP rake case with 60 local total pressure probes and six equally spaced
circumferential wall static pressure taps.
Station 4 - Fan inlet measurement plane located approximately 1 inch upstream of the
fan leading edge (~40% chord).
Station 5 - Fan outlet measurement plane located approximately 0.375" downstream
of the fan trailing edge (~23% chord).
Station 0

Figure 53: Sample Goal ID 12 - Directory of text extracted from Sample Goal ID 9

Figure 54: Sample Goal ID 13 - "Classify Finish" JSON created once classification is complete

Figure 55: Sample Goal ID 14 - "Pipeline Finish" JSON created once the last service has completed