MONASH University
MALAYSIA

FIT3162
Final Project Report

Project Title: Classification of Retrieved Documents for User Satisfaction

Team MA_A_2

Team Members:
Lim Jun Qing, 30029937
Alfons Fernaldy, 30127831
Chang Yin Cheng, 29792029

Supervisor: Dr. Prabha Rajagopal

Section	Word Count
1 Introduction	387
2 Background	544
3 Methodology	3037
4 Project Management	1,110
5 Outcomes	2000
6 Software Deliverables	1072
7 Critical Discussion	874
8 Conclusion	233
Total	9257

Contents

1 Introduction

This project aims to address the lack of readability consideration in modern retrieval systems by developing a classifier capable of predicting a document's readability based on the user's comprehension level.

Information retrieval systems, commonly known as search engines, represent the gateway for all internet users to find and access information on the world wide web. Modern search engines retrieve information based on the relevancy of the user's search query hence we consider them as system oriented evaluation. This approach however fails to take into account the user's comprehension level especially in the English language which has remained the internet's lingua franca since the 1960s (Danet, B., Herring, S. C., & Herring, S. C, 2018). Now with over 4.66 billion internet users worldwide and growing, the difficulties of non-proficient english speakers needs to be considered when assessing retrieved information based on system oriented evaluation search engines.

An alternative evaluation method proposed involves incorporating a more user-oriented evaluation to search engines by taking into account the difficulty of retrieved documents. A recent study found that effort is an important factor in determining document utility (Verma, Yilmaz, Craswell 2016). Traditionally, document readability has always been measured through a myriad of formulas such as the Automated Readability Index (ARI) or the Coleman-Liau Index (CLI) which wholly relies on metrics such as characters in words, words in sentences and other directly measurable text features.

The goal of this project is then to develop a readability classifier which has the ability to outperform conventional readability formulas by leveraging the BiGRU Attention neural network architecture. A full stack system consisting of a web application and server will also be developed to serve as the project foundation and proof of concept. The server will be used to host the model and the web application would allow non-technical users to submit documents and view readability classification results conveniently.

For future developments, our classifier can be hypothetically integrated into a search engine in the form of a third-party browser extension or by direct integration from search engine providers. Therefore if developed sufficiently, it is believed that the fully realized outcome of this project is able to positively impact the user experience of millions of internet users worldwide allowing those with limited english proficiency to access both relevant and readable resources and documents on the world wide web.

2 Background

Readability is the relation between a given text and the cognitive load of a reader to comprehend or understand it. Numerous readability formulas have been developed to measure the readability level of a reader. These formulas focus on the usage of lexical and syntactic features with various statistical factors such as word length, sentence length and difficulty of these words to calculate the readability level (Martinc & Robnik, 2019; Davison & Kantor, 1982). Traditional methods of predicting readability purely with statistical correlations have been criticized for its lack of accuracy and a lack of validity on the prediction results (Crossley, 2017).

As such, multiple research efforts have been conducted to develop new approaches to measure readability. One such method considers measurement as a classification task that predicts the readability scores using a set of attributes from the texts (Pantula & Kuppusamy, 2020). These approaches are adaptable because it yields better and accurate results in measuring the scores for different text files. However, these methods require additional resources such as labeled readability datasets which are scarce.

The rise of neural networks and machine learning has made text classification with the use of text-features a possibility. Many researchers have found convolutional networks or deep neural networks to be useful in extracting information from raw signals, ranging from computer vision applications to speech recognition and others (Zhang & Zhao & LeCun, 2015). These are shown to have impressive performance on numerous language related tasks. In fact, using deep neural networks to perform classification on texts have shown competitive results when a large scale of datasets are used (Zhang & Zhao & LeCun, 2015).

In this literature review, we will review the performance and application of neural networks, particularly on the network of Bidirectional Gated Recurrent Unit used for this task. This will provide us an accurate perspective over the use of deep learning techniques to achieve text classification for predicting document readability.

2.1 Literature Review

Summary of Previous Literature Review

Literature	Description
Martinc, M., Pollak, S., & Robnik-Šikonja, M. (2019). Supervised and unsupervised neural approaches to text readability. *arXiv preprint arXiv:1907.11779.*	Recurrent Neural Networks (RNN)
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.	Long Short-Term Memory (LSTM)
Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*	Bidirectional Encoder Representations from Transformers (BERT)
Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016, June). Hierarchical attention networks for document classification. *In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies* (pp. 1480-1489). Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.	Hierarchical Attention Network (HAN)
Sun, Y., Chen, K., Sun, L., & Hu, C. (2020, July). Attention-based Deep Learning Model for Text Readability Evaluation. *In 2020 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.* Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271.*	Bidirectional GRU (BiGRU) with Attention Mechanism

(Table 2.1: Summary of Literature Review from Proposal)

The table above summarizes the literature review that was accomplished in the previous project proposal. According to Mikolov et al. (2011), neural language models are able to outperform n-gram language models by a high margin with relatively small data sets (less than 1 million tokens). The use of neural networks in determining readability was heavily researched and it was discovered that neural classifiers outperform the state-of-the-art standard approach via readability formulas. Particularly, the BiGRU and HAN classifier managed to obtain the readability level of a text with high accuracy. As such, the following literature review will focus on the discussion over the usage of BiGRU and HAN.

Bidirectional GRU (BiGRU)

This deep reinforcement learning model is enhanced with the use of an attention based bidirectional-GRU (Gated Recurrent Unit, which is a simplified version of Long Short-Term Memory cell) model (Sun, Y. Chen & Sun, L. & Hu, C., 2020), as illustrated in the figure 2.8 below.

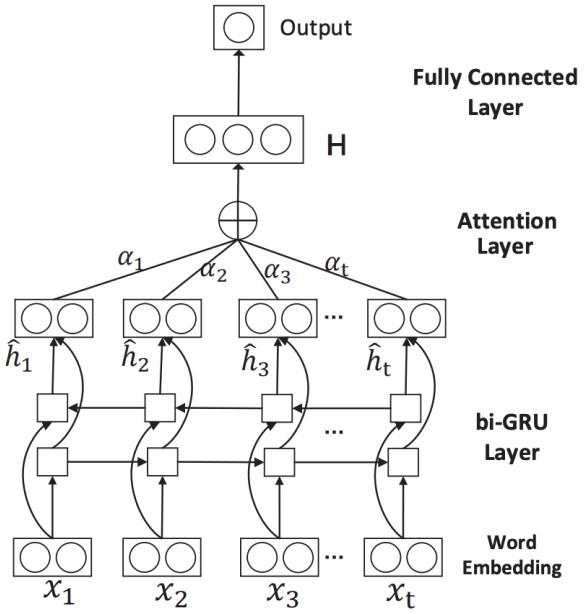


Figure 2.1: Illustration of Attention-based bi-GRU Model

In the model above, H is fed to a fully connected layer and then a softmax layer that outputs the probability distribution $p = (p_1, \ldots, p_l)$ of the readability levels. Subsequently, for a sentence, the readability level (RLs) is then defined as the label whose probability is the maximum in p, For a passage or document, the readability level is defined as the average of RLs of all sentences. As such, this helps in enhancing the accuracy of document readability (Mohammadi, H. & Khasteh, S., 2019).

Hierarchical Attention Network (HAN)

Hierarchical attention network (HAN) is an attention mechanism in the neural network proposed by Yang et al (2016). It exploits the hierarchical nature / structure of the text data and adopts the attention mechanism for classifying documents (Bahdanau, Cho, & Bengio 2014).

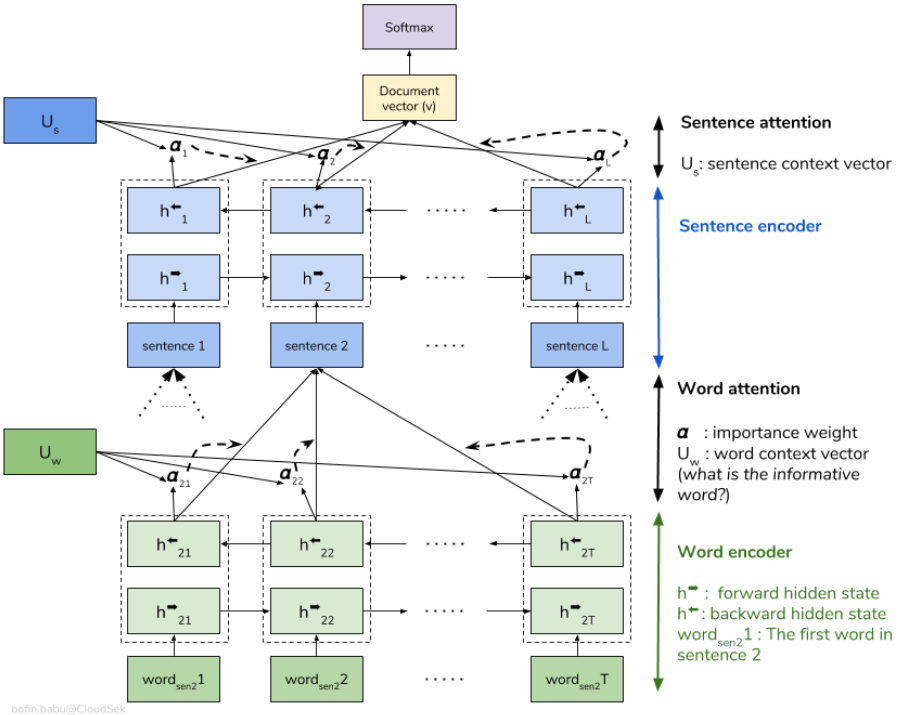


Figure 2.2: Illustration of HAN Model

The basic idea of this architecture is that this is a double layered structure of the BiLSTM model with Attention Layer on the top of each layer. The lower part of the architecture , indicated in green, is trained on each individual word embedding vectors in each sentence parallely, that is, for each sentence, their output in the lower part will be different. We call this part of the model as Word Encoder since it encodes the individual words. Then, the output vector from each sentence will be concatenated into a matrix which will have a dimension of (num_sentences, vector_length). This matrix will then be fed into the upper part of the model which is indicated in blue, we call it as Sentence Encoder since it encodes the vector processed from each sentence. The output of Sentence Encoder will then be fed into a fully connected layer and perform classification to the input batch of texts.

2.2 Summary

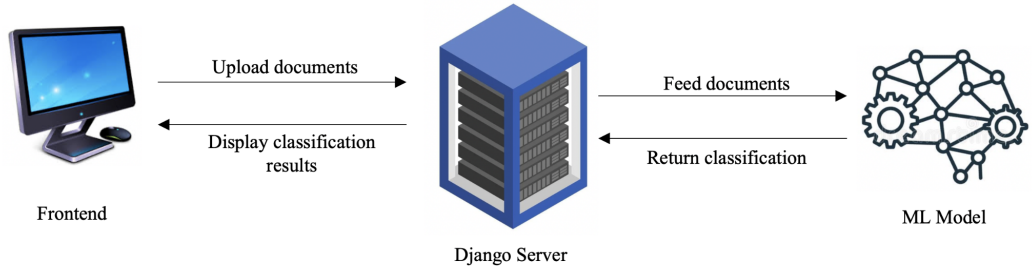
Based on our findings, we discovered that there are many different methods for determining readability levels ranging from formulas to neutral networks. We explored the use of neural networks in determining readability and discovered that neural classifiers outperform the state-of-the-art standard approach. Particularly, the HAN and BiGRU based classifiers managed to obtain the readability level of a text with high accuracy. As such, we decided to apply BiGRU and HAN in determining the readability level. The implementation details will subsequently be discussed in the Methodology section.

3 Methodology

The main purpose of this project is to build a classification model to classify user documents on readability level. As such, the following sections discuss the components or the methodology used to achieve the end goal, which consist of three major parts, as outlined in the diagram below.

(Figure 3.1: Project methodology flow)

3.1 Overall System Architecture



(Figure 3.2: overall system architecture)

The overall system architecture contains three major components. Figure 3.2 shows the interactions between these components. The frontend component displays the user interface for the user to upload documents for classification. The backend (Django server) component receives user inputs and performs computations on it. Finally the machine learning model receives inputs passed in by the server and returns the classification results.

This integrated system of the three modules described above is designed according to the Model-View-Controller to separate the different aspects of the project (Krasner & Pope, 1988). It creates a loosely coupled set of modules which simplifies feature testing and reduces the risk of creating a domino effect when a component of the system fails to operate. Each of these modules is then operated independently but integrated into the system as a whole. The following discusses these components.

Frontend (User Interface)

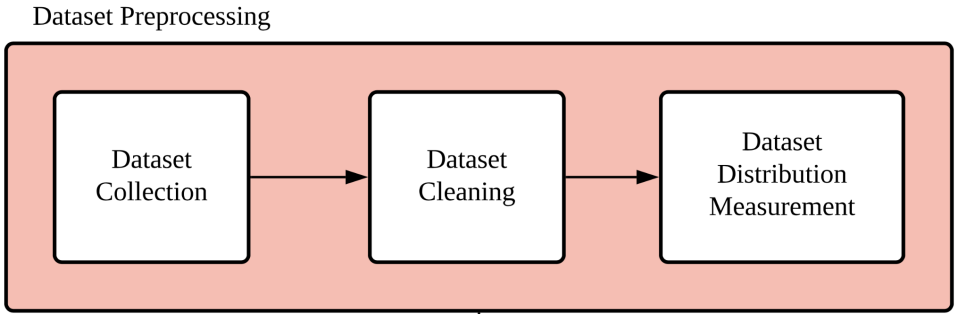
The front-end application was built using ReactJS. Its main purpose is to provide the user access to the classification model by uploading documents to be classified via a user-friendly dashboard. The dashboard will then send the document(s) to the backend server to be preprocessed and classified. When the classifier completes the classification process, it sends the results to the front-end with HTTP protocols. The front-end will then display the readability levels of the documents uploaded.

Backend (Server)
The back-end server was built using Django. It serves as an intermediary between the front-end and the machine learning model. This server is responsible for receiving documents from the user frontend via API endpoint calls. When the server starts, it connects automatically with the machine learning model to speed up the classification process. Documents that are received by the server will be preprocessed allowing the text content to be extracted and fed to the machine learning model for classification. When the classification result returns, they are sent back to the front-end in the form of a JsonResponse. The frontend parses the JSON and displays them to the user accordingly.

Machine Learning Model (Classification Model)
The Machine learning model represents the main product of the entire project. It is a classification model that categorizes documents into discrete readability categories. The classification model was developed using popular libraries such as TensorFlow, Scikit-Learn, NLTK, pickle, numpy, custom data manager, etc. After the classification model is trained, it will be pickled and hosted onto the back-end server.

Several classification techniques are used to build different readability models for the purpose of performance comparison. The best performing model is determined based on the analysis of evaluation metrics such as accuracy. This model is subsequently used as our final model hosted on the server. The details of the implementation are described in Section 3.3 below.

3.2 Dataset Preprocessing



(Figure 3.3 Data preprocessing steps)

The choice of datasets had an influential impact on the overall performance of the classification model (Holte, 1993). An accurate classification model can only be achieved when the dataset used to train it is properly categorized based on their readability level. Doing so is challenging at best as readability is a metric that is highly debatable. Therefore a significant portion of this project involved formulating a proper methodology to determine the actual readability levels of each extracted dataset.

Figure 3.3 above illustrates the steps that were taken to collect, clean, and standardize datasets. As our project involves a substantial amount of corpora dataset, the distribution levels of each dataset for each readability level (from elementary to undergraduate) were calculated with the help of a python script. The distribution levels of each data source are shown in the table below.

Datasource	Elementary	Middle School	High School	Undergraduate
ALevels - English Literature	5.56%	44.44%	30.56%	19.44%
IELTS Reading	0.00%	5.00%	70.00%	20.00%
Monash Library	0.00%	0.00%	3.88%	96.12%
EReadingWorksheet	0.00%	93.75%	6.25%	0.00%
British Council	39.47%	57.89%	2.63%	0.0%
ISL Collective (beginner)	20.51%	48.21%	10.77%	18.97%
ISL Collective (elementary)	13.02%	44.88%	17.32%	23.14%
ISL Collective (advanced)	6.12%	40.44%	23.34%	27.56%
OneStop (elementary)	0.00%	28.04%	46.56%	25.4%
OneStop (intermediate)	0.53%	24.34%	61.38%	13.76%
OneStop (advanced)	0.00%	3.17%	25.93%	70.9%
WeeBit (BitGCSE)	7.32%	56.38%	31.50%	4.80%

WeeBit (BitKS3)	0.00%	100.0%	0.00%	0.00%
WeeBit (WRLevel2)	21.78%	74.44%	1.24%	2.54%
WeeBit (WRLevel3)	7.12%	88.01%	4.49%	0.37%
WeeBit (WRLevel4)	9.50%	83.49%	6.58%	0.43%

(Table 3.1: Dataset distribution level comparisons)

Note: The sources for the above data sets are provided in the Reference section

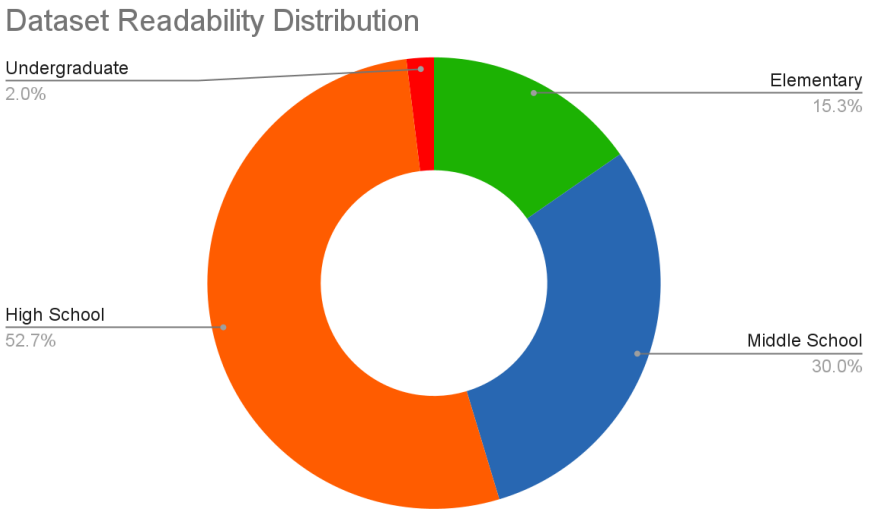
This script uses the textstat library which provides ready-made functions to calculate the readability of extracted text. In particular, we used its aggregation function to calculate the average readability level of a document across different readability formulas. Running the script generates a readability test report text file containing the readability level for each document as well as the distribution of readability levels (Table 3.1). Using the spread, we are able to determine the overall readability level of each dataset and categorize them appropriately. Categorized datasets are then combined into one single large dataset which we used as training data for the classification model. A code inspection on this script can be found in Appendix A: Data Preprocessing Source Code.

Analysis of Datasets

After obtaining the final combined dataset, we can tally up the number of documents obtained for each category to check for any skew. These are shown and visualized below:

Level	Number of Documents
Elementary	2289
Middle School	4881
High School	7869
Undergraduate	294
Total	15,333

(Table 3.2: the combined statistics of all datasets)



(Figure 3.3 Pie chart of all datasets)

From the table and pie chart, we can observe that there is a large amount of skew in the dataset distribution. Each category should ideally make up 25% of the dataset but in this case there is an abundance of High School and Middle School documents with Undergraduate documents being grossly underrepresented.

Undergraduate Imbalance

Undergraduate documents sampled consisted mostly of research papers and academic journals which are significantly longer than other categories' documents. We are also solely relying on the Monash library as our data source which makes it difficult to automate data collection using existing data scraping scripts. For one, each resource provided by the library is hosted on a variety of different sites each requiring the user to authenticate their institutional affiliations. The layouts of these sites are also different with some being blocked or missing. Hence, we opted to collect the data manually which is why there are only around 100 undergraduate documents.

Elementary School Imbalance

Elementary documents are very difficult to find online as most content on the internet are geared towards middle schoolers and up. After a meeting with our Project owner, Dr Prabha, we managed to discover several archival sites storing worksheets designed for elementary students. Using scraping scripts, we automated document extraction from the site but unfortunately the quantity of documents were still lacking compared to the larger corporas for Middle school and High school.

Solution for Dataset Imbalance

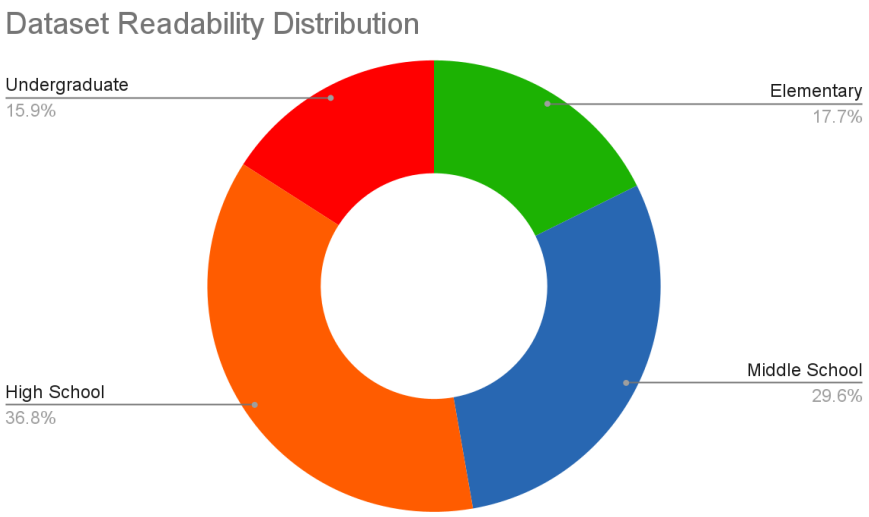
Solving the undergraduate imbalance is straightforward. Using a python script, we extracted the text of undergraduate documents which on average contains thousands of words. These are then split up into multiple smaller documents each containing 750 words or less which is the average word count for high school and middle school documents. Using this technique we were able to convert 294 documents into 2282 smaller documents.

Solving the elementary school imbalance was more difficult. As we had discussed, the issue lies in the difficulty of finding reliable data sources of elementary school documents. Re-evaluating the middle school dataset, we discovered that there were batches of documents that were labelled as elementary by the textstat library. After reading a small sample, we decided to reallocate about 241 documents into the elementary dataset.

Finally to address the high school bias, we excluded 2529 documents which had been extracted from the corporas and marked as either elementary or middle school. Ideally we want all the categories to match High school's quantity but this is just not feasible within the current time constraints. Therefore some excluding some data to flatten the distribution is something we felt was necessary,
Our final dataset spread after redistribution, splitting and exclusion are shown below:

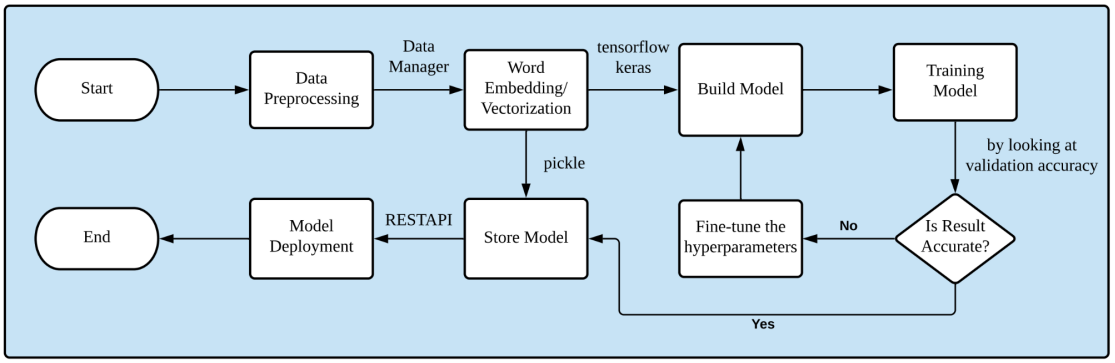
Level	Number of Documents
Elementary	2530
Middle School	4240
High School	5277
Undergraduate	2282
Total	14,329

(Table 3.4: the combined statistics of all datasets)



(Figure 3.5 Pie chart of improved datasets)

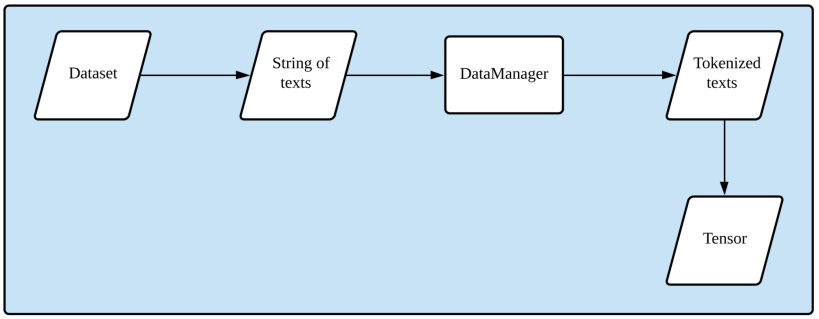
3.3 Implementation of Classification Model



(Figure 3.6: High-level overview for classification model training)

Figure 3.6 above shows the high-level overview of training and building of our classification model. Before the model can be trained with the datasets obtained from 3.2 Dataset Preprocessing, BiGRU and HAN architecture of a TensorFlow model needs to be built first. As such, the following sections illustrate and discuss the steps undertaken to build and train the model for both architectures.

3.3.1 Introduction to Conversion of Dataset to Tensors

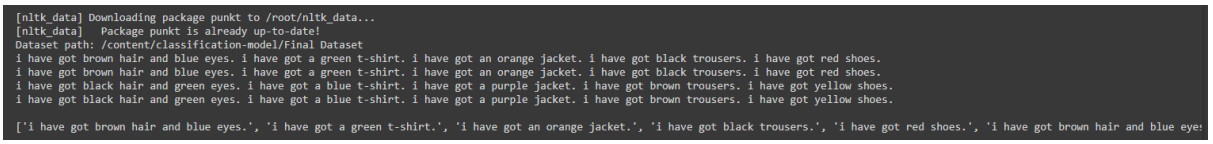


(Figure 3.7: The transformation flow of the dataset to tensors)

TensorFlow models by default implementation does not allow strings to be used as input to the tensors (Tensorflow, 2021). Hence these strings of texts from the dataset need to be preprocessed before feeding into the model for training. We define the term "DataManager" as a converter that converts or preprocesses a string of text to a set of tokenized texts in tensor format (as illustrated in Figure 3.7 above). Since our dataset comprises 4 categories (as according to Table 3.5 above), thus the output of this conversion will be 4 different tensors for the 4 different readability levels. We discuss the methodology of this process in the following sections.

3.3.2 Tokenization of Phrases

Using Figure 3.7 as illustration, the DataManager receives the input text from the dataset and uses the NLTK pretrained (default) tokenizer to split the text into individual sentences. An example is shown below.



(Figure 3.8: Tokenization of phrases)

Figure 3.8 above shows a snippet of extracted text string:

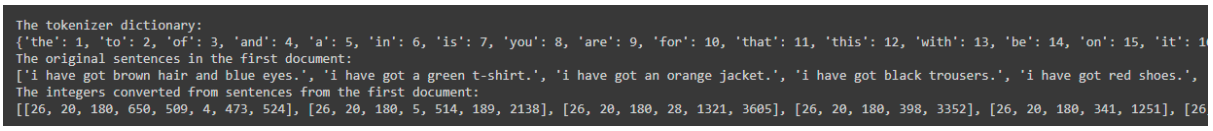
```
“i have got brown hair and blue eyes. i have got a green t-shirt. i have got an orange jacket. i have got black trousers. i have got red shoes.
i have got brown hair and blue eyes. i have got a green t-shirt. i have got an orange jacket. i have got black trousers. i have got red shoes.
i have got black hair and green eyes. i have got a blue t-shirt. i have got a purple jacket. i have got brown trousers. i have got yellow shoes.
i have got black hair and green eyes. i have got a blue t-shirt. i have got a purple jacket. i have got brown trousers. i have got yellow shoes.”
```

Running the NLTK pretrained tokenizer splits the text into an array of sentences (series of tokens) as shown below:

```
['i have got brown hair and blue eyes.', 'i have got a green t-shirt.', 'i have got an orange jacket.', 'i have got black trousers.', 'i have got red shoes.', 'i have got brown hair and blue eyes.', 'i have got a green t-shirt.', 'i have got an orange jacket.', 'i have got black trousers.', 'i have got red shoes.', 'i have got black hair and green eyes.', 'i have got a blue t-shirt.', 'i have got a purple jacket.', 'i have got brown trousers.', 'i have got yellow shoes.', 'i have got black hair and green eyes.', 'i have got a blue t-shirt.', 'i have got a purple jacket.', 'i have got brown trousers.', 'i have got yellow shoes.']
```

3.3.3 Transforming Tokenized Texts to Tensors

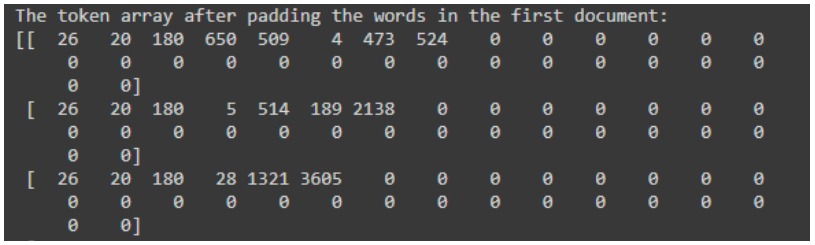
When an array of sentences is obtained, the DataManager proceeds to fit the original plain text into the TensorFlow tokenizer. This fitting process allows the tokenizer to scan the entire string to compute the number of unique words with its corresponding frequency present in the corpus (Robby, 2019). The tokenizer then produces a dictionary that maps a word to the frequency. Each string in the tokenized texts (as obtained in 3.3.2 Tokenization of Phrases), will then have its string split to map the words to an unique integer using the dictionary obtained. A screenshot is captured as below to illustrate an example of the conversion.



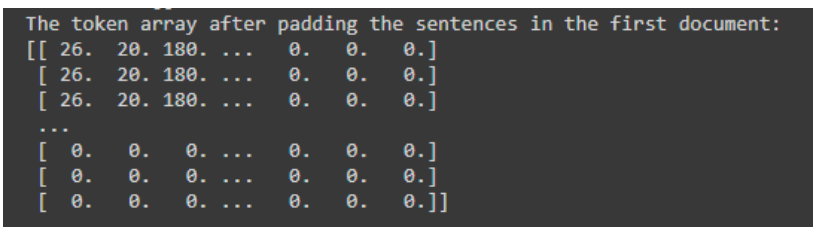
(Figure 3.9: Transformed tokens to integers)

As observed above, each string in the tokenized texts is first split to obtain its individual words, while subsequently being mapped to its corresponding integer, which is an indicator of its frequency that it appears in the entire corpus dataset.

Next, to convert the nested array of integers into TensorFlow Tensors, the array needs to be in a regular shape (i.e., a fixed length for each nested-array). After careful trial-and-error, we discovered that a sentence length of 30 words to be the most balanced length. As such, each sentence with a length < 30 will be padded with an integer of 0, while a length > 30 will have its sentence truncated into exactly 30 words. Apart from sentence trimming, the number of sentences will also be trimmed to 100, such that the preprocessor only takes in the first 100 sentences of each document. This produces the following output:



(Figure 3.10: Padding of 0s to each sentence)



(Figure 3.11: Finalized padding of each sentence)

When the padding process is completed, the tensor will have a shape of `(num_of_documents, 100, 30)` where `num_of_documents` is the number of documents in the dataset, '100' refers to the first 100 sentences of each document while '30' refers to a fixed size of 30 words for each sentence.

3.3.4 Mapping Tensor with its Readability Level Class

After the DataManger obtains the tensor, the next task is to map the tensor with its original readability level as defined in Table 3.5 above. This provides us an accurate dataset of tensors for each category of readability level. Namely, we encode the tensors into the following categories:

- 'Elementary School' is encoded as 0
- 'Middle School' is encoded as 1
- 'High School' is encoded as 2
- 'Undergraduate' is encoded as 3

The DataManger will then shuffle the tensor with the corresponding encoded labels as defined above. Once completed, it splits the tensors into a training-validation ratio of 0.8 (where 80% of the tensor belongs to the training set while 20% belongs to the validation set). This results in the tensors to have the following shapes:

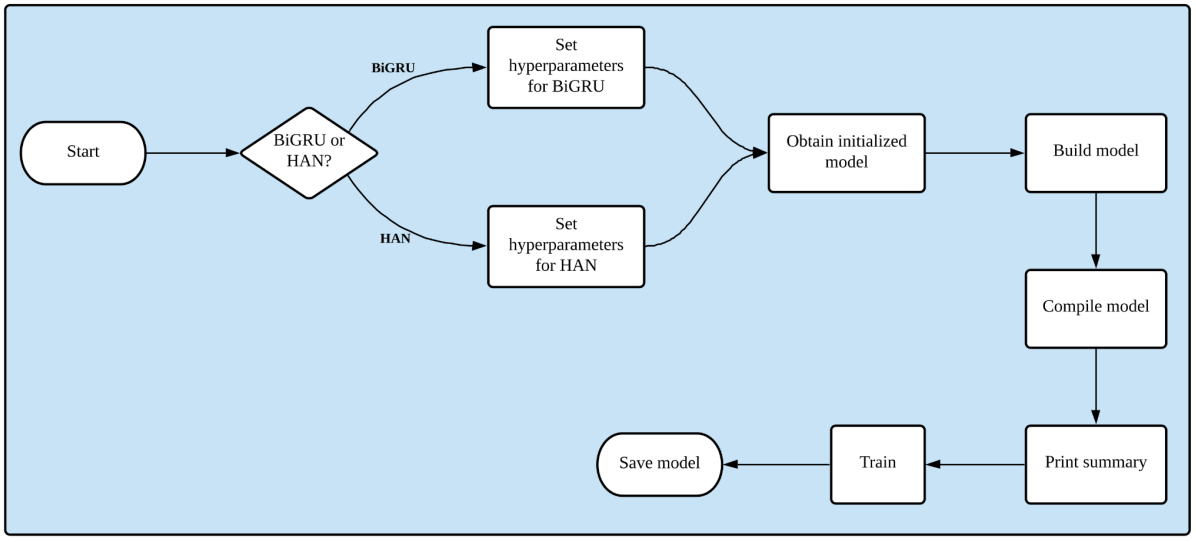
```
Shape of training set and the length of its label vector:
(12265, 100, 30) 12265
Shape of validation set and the length of its label vector:
(3065, 100, 30) 3065
```

(Figure 3.12: The shape of training and validation set)

Having the training set and validation set to be separate sets allow us to perform model training (which will be discussed in subsequent sections) and model fine-tuning with hyperparameters. When finalized, it produces 4 different tensors for each of the readability level categories as defined in the encoding above. These 4 tensors are the dataset corpus, whose words are tokenized, transformed and mapped with its corresponding readability level.

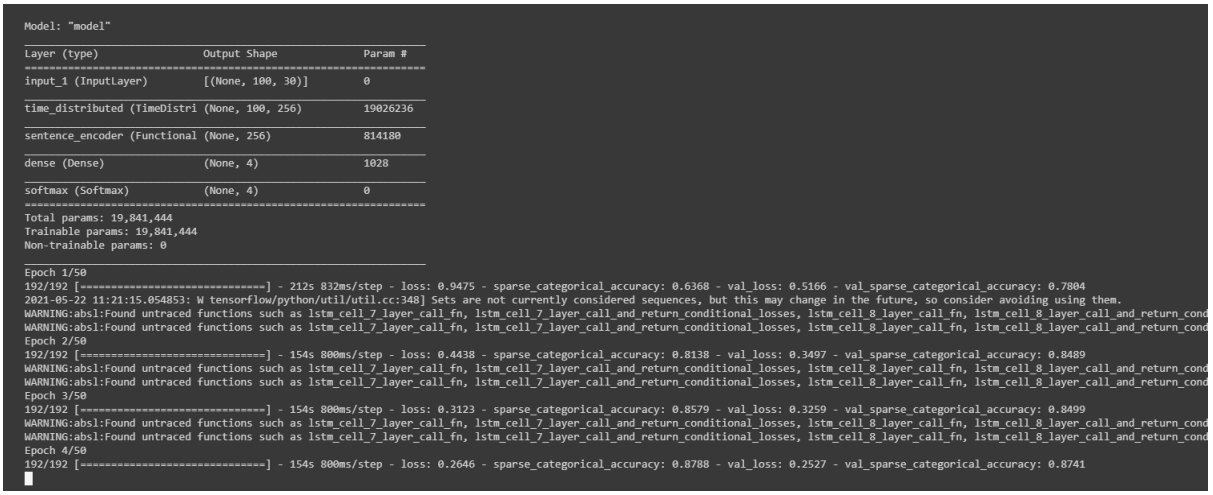
3.3.5 Building and Training of Classification Model

After the DataManager obtains the 4 tensors, a training script is written to build the model according to the architectures selected in Section 2.1 Literature Review, namely BiGRU and HAN architecture. A flowchart of the training process is illustrated as below.



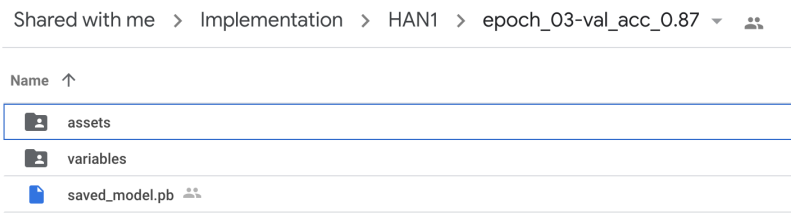
(Figure 3.13: Flowchart of classification model training)

Once the model of the corresponding architecture with its hyperparameters is set, the model is built using our written script, compiled and has its summary printed out to the display for documenting and recording purposes, while continue with training of the model (using the training set of the tensor obtained from the former discussion).



(Figure 3.14: Output display of the training process)

Figure 3.14 above illustrates a sample of the summary printed out (as a table) upon the training of the model. The table represents the summary of the HAN model with the details of the number of parameters used to train, etc. When the training completes, the model will be stored into different folders indicated with the number of epochs and the validation accuracy.

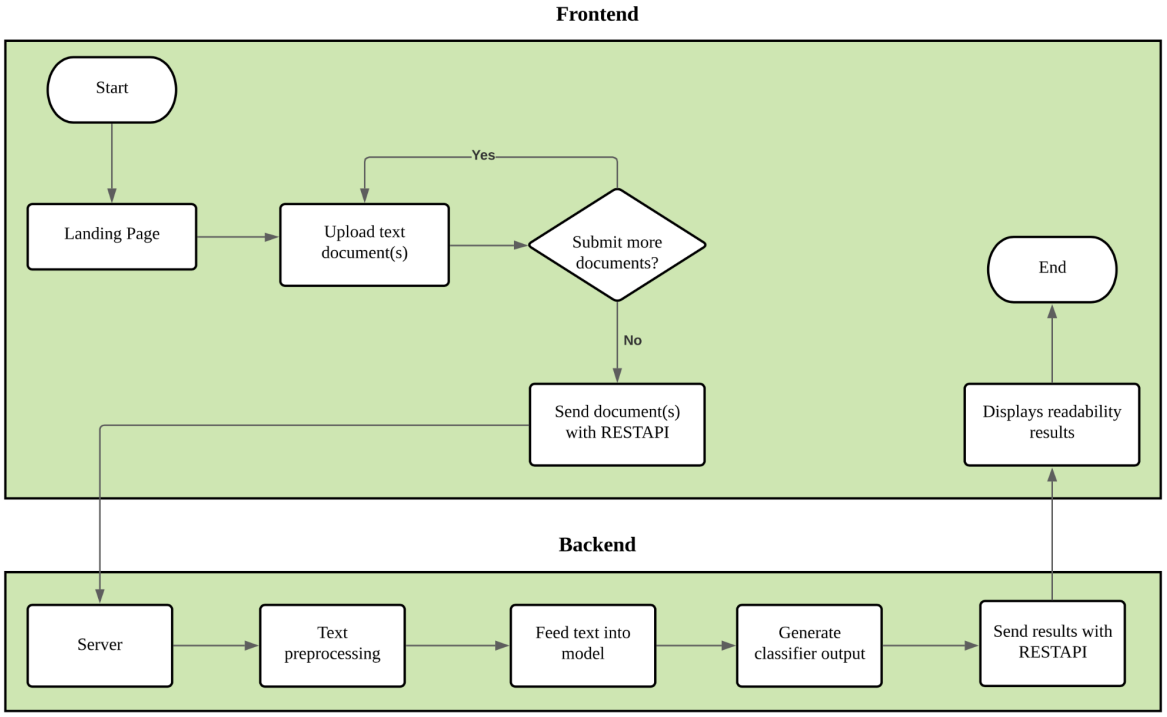


(Figure 3.15: A sample model trained with Google Collab and stored on Google Drive)

The accuracy and the error or loss of the model will then be shown in every epoch during the training process. The performance of the model can then be judged by looking at the validation accuracy (where it is validated during the training process). If the validation accuracy is not improving, different hyperparameters were fine-tuned again to obtain different performance results. This is in accordance with the recommended fine-tuning approach as described in Bonaccorso, 2018. As such, different hyperparameters of the architecture were used in the training process. These results were then documented accordingly and the model with highest accuracy score was chosen as our final classification model. When the final model is chosen, it is then saved and ready to be deployed to the Backend Server. A deeper analysis on the results of variations model trained is discussed in the 5.4 Discussion of All Results section.

3.4 Implementation of Web Application

After developing the classification model, the next step involved creating a web interface for users to be able to access the deployed model.



(Figure 3.16: Interaction between the frontend and backend process)

Frontend Application

The frontend web application was constructed using React - a JavaScript framework for building user interfaces. State management libraries such as Redux and Redux Saga are used to manage global state and handle any asynchronous HTTP calls to the backend server. Redux uses a single store instead of many different stores like other flux inspired libraries do therefore is often called ”single source of truth”, as the store is the only place where state can be accessed from (Caspers M, 2017). Meanwhile React Dropzone is also used to allow users to upload and review documents before they are sent to the server as well as handle any edge cases. These include exceeding the document limit, uploading an unsupported document type or uploading the same document again. Regardless of the error the web application catches them and makes it clear to the user by displaying either a modal or a toast with the error’s description.

After documents are uploaded to the server, the web application will get back a response containing the results of classification and subsequently display the results of each document to the user in a result screen.

Web Design Incorporated

Our web application design is inspired by modern minimalist design which puts emphasis on the use of white space, better typography, grid layouts and less colours to express content. Minimalism follows the “less is more” philosophy wherein only absolutely necessary graphic and content elements should be included (Meyer, 2015).

The following design principles were incorporated into our design:

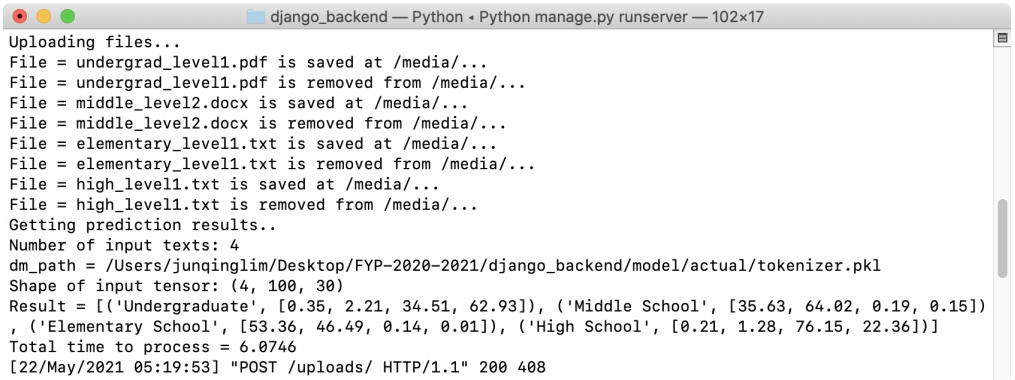
1. Visual Hierarchy: arrangement of elements based on their order of importance
2. Navigation: simple, intuitive and consistent navigation
3. Simplicity: fewer components are used to achieve more
4. Content: the use of compelling and non-ambiguous language to cater to users

Our team strongly believes that good user experience is a key factor when developing a product for users. Poor UI/UX can negatively impact user engagement. No user would appreciate long load times, cluttered UI or convoluted navigation. In the real world, these factors can make or break projects which is an aspect which cannot be overlooked.

Backend Server

Our initial plan was to use Flask for our backend system. However, subsequent iterations of development found that it was more useful to use Django instead as the project was possible to be scaled up. The backend server was built with Django to handle the aspect of classification model loading and file uploading. The classification models built are stored as TensorFlow (SavedModel format) and tokenizer is stored as a stream of bytes using Python's pickle library (i.e., stored as .pkl file). As the classification model takes a few minutes to load, it is loaded alongside the server. This reduces the time taken to obtain the predicted results as we don't have to keep reloading the server for each request.

On the aspect of file uploading, Django's FileSystemStorage is used to perform the action of storing the files. Files uploaded were also deleted upon the completion of text extraction to ensure the compliance of user privacy and confidentiality. To extract the text from different file formats (pdf, txt and docx), the textract python library is used. This library streamlines the extraction of text from multiple files with different file formats. Django then sends these extracted text to the pickled model for further processing and classification. Results are then returned to the frontend with its API call to be displayed to the user.



(Figure 3.17: Process of uploading files to removing the files from storage and lastly obtaining the results from the model)

All computations performed ranging from loading the classification model, cleaning text and classifying documents are performed directly on the Django server.

4 Project Management

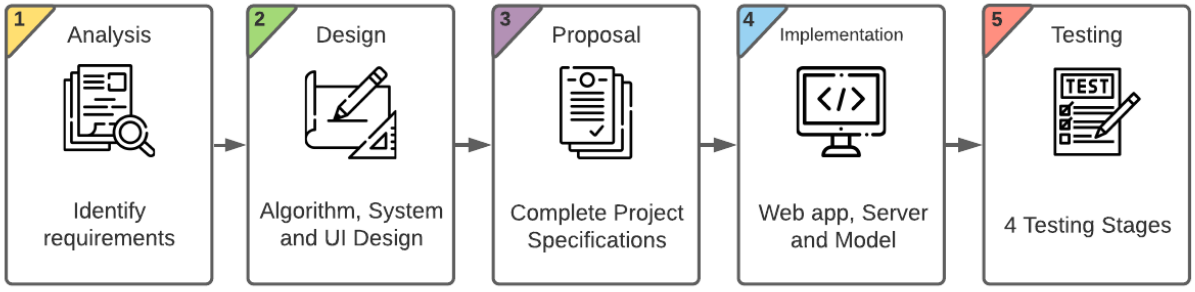
4.1 Brief Intro to Project Management

During the lifecycle of this project, its management has closely adhered to the original project proposal that was prepared in the previous semester. Fundamental steps such as deciding on a process model, selecting the appropriate project management tools and allocating tasks between members were executed according to the proposal. Kanban boards Gantt charts, Work Breakdown agreements and Risk registers served as our primary project management tools (refer to Appendix). Their usage allowed us to firmly establish a clear project timeline, highlight key milestones and manage unforeseen circumstances during the project's lifecycle.

4.2 Our Approach (Process Model)

Projects generally adopt one of two process models, Agile or Waterfall (Predictive). Agile is an iterative model with emphasis on incremental and rapid development cycles which build on existing functionality. Each iteration is tested thoroughly to ensure that the quality of the deliverables are maintained. All of this makes Agile a popular choice for software based projects which are oftentimes time-sensitive and require flexibility to accommodate the needs of the client.

On the contrary we have the traditional Waterfall model which is sometimes referred as the Predictive model. Waterfall model provides a series of well defined sequential steps which must be traversed linearly. What this means is that each phase of a project must be fully completed before the next one can begin (Bassil, 2012). Reviews are conducted primarily at the end of each phase and the team must decide whether they are ready to proceed to the next phase. Compared with Agile, the Waterfall model is quite simple to use with clearly defined deliverables for each phase. There is very little overlap between phases making it easier to manage and review.



Waterfall Model's 5 Phases

For our project, the team selected the more rigid Waterfall model, given the amount of preparations and planning that were done in the previous semester. The Waterfall model initially worked well for us as this was a small project with supposedly well defined deliverables. However during the later stages of the project we started to shift more towards

the Agile methodology when requirements changed unexpectedly. For example, due to poor initial performance, we were forced to switch our neural network architecture which made some of our plans obsolete. In such scenarios, Agile's rapid development cycles clearly had an upper hand as it allowed us to quickly adapt based on continuous evaluation which is commonplace when dealing with software based projects.

4.3 Project Resources, Execution Management and Planning

Project Resources

Project resources represent the people, capital and material goods required for the successful execution and completion of a project (Elizabeth, H, 2016).

The team represents the project's most invaluable resource and is the first we shall highlight. Each member of the team is assigned a preordained role within the project shown below.

Team Member	Role	Responsibilities
Lim Jun Qing	Project Manager	The Project Manager coordinates with every member and ensures the smooth running of the project. They closely monitor the project's progress to ensure that milestones are met within the set deadline and that each team member fulfills their assigned tasks within their allotted time period.
Yin Cheng Chang	Technical Lead	The Technical Lead spearheads the design and implementation efforts for the technical deliverables of the project. They ensure that the software being developed meets all the project's requirements and has the authority to make major technical amendments to the project.
Alfons Fernaldy	Quality Assurance	The Quality Assurance ensures that the quality of each deliverable whether it be small incremental additions or entire modules are up to standard. This means performing code reviews and conducting unit, system and integration testing with module owners.
Dr. Prabha Rajagopal	Project Owner	The Project Owner has extensive knowledge on the field covered by the project. As a result they guide the direction of the project and serve to provide advice and feedback on the progress the team has achieved.

Other project resources which are required represent our hardware and software needs. These are very well documented in the project proposal with no amendments necessary.

Execution Management
As mentioned above, the execution of a project is solely managed by the Project Manager. While the team is busy designing and constructing deliverables based on the project plan, the PM tracks the project's progress and ensures milestones and deliverables are aligned with the project's schedule.

All communications were monitored by the PM and ideas proposed are discussed openly. This keeps the PM constantly updated with the current status of the project as well as any ideas the Technical lead or Quality assurance would like to implement. For example, when we discovered the lackluster performance of our first developed model architecture, the PM was able to organize an impromptu meeting to discuss the immediate steps going forward.

Planning
Project planning involves the creation of project management artefacts which document both the schedule and scope of the project. Before starting our project we had identified both Functional and Non functional Product requirements which our project deliverable needed to satisfy. To complement this we also wrote a Product User Acceptance Criteria which are the conditions for our software to comply with. Combined with our well defined scope, we were able to plan out all the requirements needed for the project to succeed.

4.4 Risk management

All projects must contend with a great variety of risks with varying impacts. Risk management therefore outlines the immediate steps the team must take to minimize the negative impacts of such risks if they do happen. More importantly it makes the team critically contemplate on the "what ifs" scenarios that can realistically occur during the project lifecycle (Raz, Shenhar & Dvir, 2002).

Our risk management process consists of 2 steps, risk assessment and risk control. Risk assessment focuses on the identification, analysis and prioritization of risks whilst risk control comes up with solutions to mitigate, transfer or even avoid the risk.

As part of the planning process, the team created a Risk register which documents and categorizes all the possible risks we could brainstorm. Each entry of the risk register therefore is comprised of the following sections:

Section	Definition
Risk description	Brief description of the risk which details how it can negatively impact the project.
Causes or Triggers	Events that would cause the risk to become realized.
Risk owners	The member who is accountable for managing the risk and taking responsibility if the risk is realized.
Risk responses	Potential responses that can be done to resolve the risk.

Risk category	The overall category of the risk (Technical, Scope, Schedule)
Probability, Impact ratings	A rating between 0-10 which records the current team sentiment in regards to the probability of the risk happening and the severity of the risk.
Risk rating	The product of the probability and impact score. Allows us to compare and prioritize risks that have high impact and are likely to happn.
Risk Rationale	Additional remarks the team had on the risk

During project execution, the risk register greatly helped in mitigating the impact of some risks that did end up happening. For example, when it became apparent that the current neural network was performing poorly, we were able to come up with a swift response to mitigate the risk by referencing our risk register. To reiterate the above point, a risk register makes us seriously consider the “what ifs” of the project so that when it does happen, we as a team would be one step ahead to handling and resolving the issue.

4.5 Limitations Encountered During Project Management

4.5.1 Geographical Limitations

The team initially had the expectation of meeting face to face for the development of the final project however the extension of Monash’s online semester prevented us from doing so. This was somewhat disappointing because we felt collaborating in person would give us more motivation as compared to doing our tasks alone. Of course we could hold online meetings and collaborations but this would not have the same feel as the traditional meetup sprints we used to have on campus. To put it in short, we believe that our isolationist workspace makes it difficult to both manage and execute the project.

4.5.2 Human Limitations

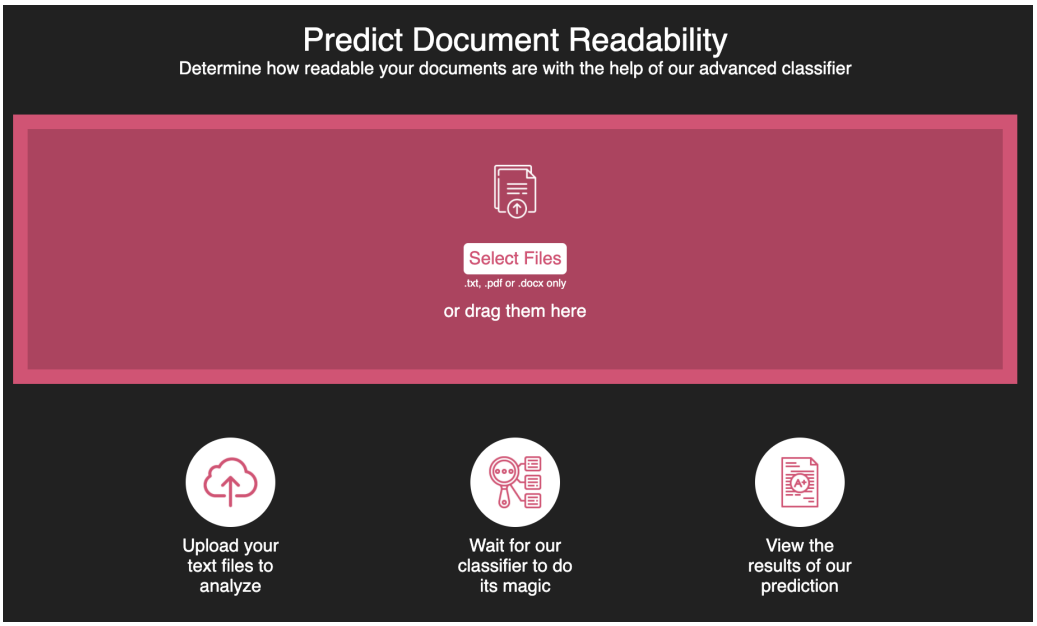
Despite the best efforts of the project manager, it is sometimes difficult to prioritize putting work into the project when each member of the team is preoccupied with dealing with other assignments. Decreasing motivation also negatively impacted the workflow and oftentimes there were extended periods where no progress was made. Of course this is to be expected because we as students will always prioritize projects whose deadlines are fast approaching and tend to put aside projects whose “hard” deadlines come much later.

5 Outcomes

This section discusses the outcomes of the project.

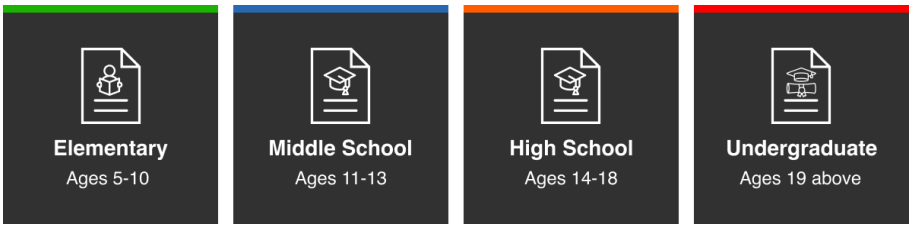
5.1 Results Achieved/Product Delivered

This project has managed to produce a readability classifier using the BiGRU Attention neural network architecture. The final product delivered is a web application which allows the user to upload documents to the server. The server then returns the predicted scores for each readability level of a document.



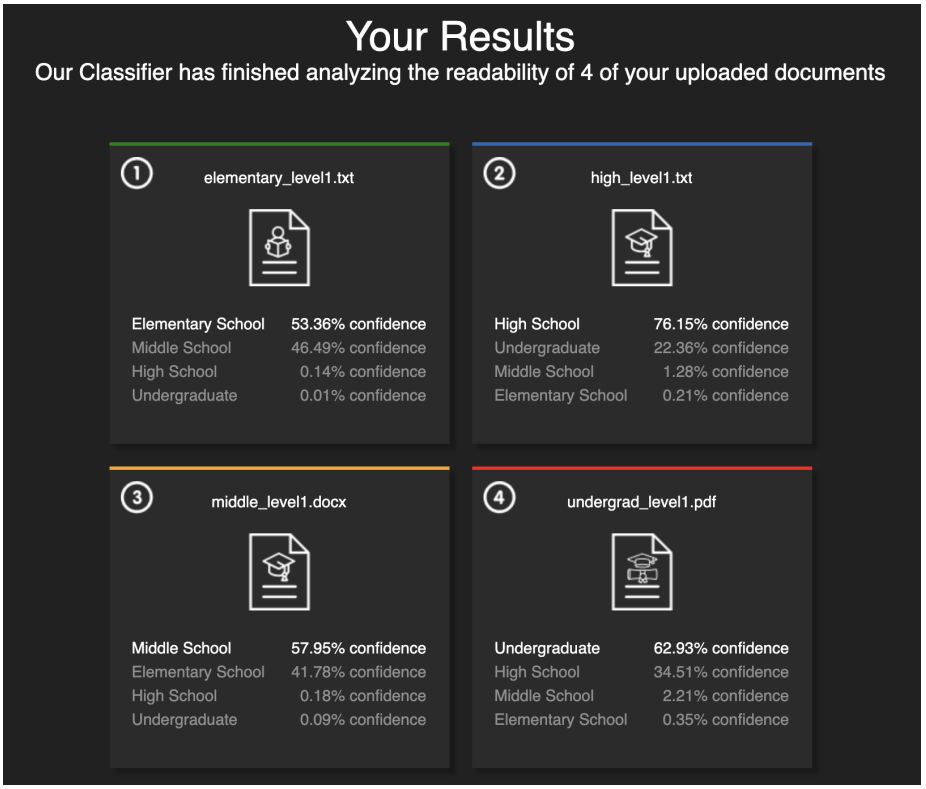
(Figure 5.1: A sample screenshot of the working product deliverable)

The model is trained such that it is able to categorize a document into 4 distinct educational categories shown below.



(Figure 5.2: Categorization of documents)

The classifier takes one or more text documents as input and outputs an array containing the percentage of confidence it assigns for each readability level. The category with the highest assigned confidence hence represents the classifier's predicted readability level as shown below.



(Figure 5.3: Results for different documents)

In addition to the readability classifier, a full stack system was also developed to host the classifier, as shown in Figure 5.1. Users can utilize the fully fledged, functional front-end web application to upload their documents and see the readability predictions made by our classifier. Meanwhile, our backend server is responsible for hosting the model capable of communicating with the application through API endpoints which support multi type document transfers.

Overall, we have achieved our goals in developing a software architecture which can serve as the base foundation for future expansions.

5.2 How Are Requirements Met

Each requirement of the project from the initial proposal planning was generalized with the following sections discussing the adherence of the requirements.

5.2.1 Dataset Construction

A classifier's performance is oftentimes determined by the quality of its underlying dataset. Therefore to produce a decent classifier, text documents were collected and categorized into the aforementioned 4 readability categories accurately. Dataset across different corpora were collected, aggregated and cleaned to be used as the training dataset of the model.

5.2.2 Readability Classifier

After finalizing our dataset, we proceeded to train our classification model using Google Colab. Training was quite straightforward as we only had to try out different configurations of neural network architectures with varying hyperparameters. After several days of training we managed to obtain a model with an 87% categorical accuracy.

```
Epoch 9/50
192/192 [==============================] - 264s 1s/step - loss: 0.2550 - sparse_categorical_accuracy: 0.8577 - val_loss: 0.4020 - val_sparse_categorical_accuracy: 0.8533
Epoch 10/50
192/192 [==============================] - 263s 1s/step - loss: 0.2326 - sparse_categorical_accuracy: 0.8831 - val_loss: 0.3795 - val_sparse_categorical_accuracy: 0.8640
Epoch 11/50
192/192 [==============================] - 264s 1s/step - loss: 0.2061 - sparse_categorical_accuracy: 0.9000 - val_loss: 0.3566 - val_sparse_categorical_accuracy: 0.8703
Epoch 12/50
192/192 [==============================] - 263s 1s/step - loss: 0.2261 - sparse_categorical_accuracy: 0.8985 - val_loss: 0.3365 - val_sparse_categorical_accuracy: 0.8688
Epoch 13/50
192/192 [==============================] - 263s 1s/step - loss: 0.2303 - sparse_categorical_accuracy: 0.8941 - val_loss: 0.3603 - val_sparse_categorical_accuracy: 0.8653
Epoch 14/50
192/192 [==============================] - ETA: 0s - loss: 0.1896 - sparse_categorical_accuracy: 0.9077
```

(Figure 5.4: Model Training in Google Colab)

We used 50 epochs to train the model but in reality there are diminishing results beyond 15 epochs and higher. In Figure 5.4, we obtained the categorical accuracy peaking at 87% during Epoch 13/50 and dropping after subsequent epochs. Hence to preserve Google Colab's limited runtime, we stop the training manually and re-evaluate our results.

5.2.3 Web Application

A website was successfully built for readability level predictions on various documents. Model is deployed with Django as the back end and React as the front end. Both functional and non-functional requirements of the website such as displaying the results are met. Usability principles such as Norman's Principles (Norman & Nielsen, 2010) and Schneiderman's Golden Rules (Euphemia, 2020) were considered upon designing the user interface.

5.3 Justification of Decisions Made

Several important decisions were made throughout the project implementation to ensure a successful project execution. The following sections critically discusses the major decisions we had made throughout the project's implementation.

5.3.1 Categorizing Readability Levels

We decided to only categorize documents into 4 distinct readability levels because of dataset limitations. Readability is a metric that is difficult to categorize precisely. The difference between a document for 10-11 and 12-13 age groups would be difficult to discern manually especially in high numbers thus making categorizing them difficult.

In contrast, one can easily tell the difference in readability between an academic journal written by an undergraduate student and an IGCSE essay designed for middle school students. This clear distinction between the 4 readability levels we have chosen would make it easier to accurately categorize documents we had found online in order to construct our final dataset.

5.3.2 Evaluating Dataset Quality

Our dataset consists of documents from two sources, namely curated and uncurated data sources. Curated sources represent those who implicitly categorize their own documents to fit our readability levels. For example, IELTS and TOEFL English exams are designed for high school students and academic journals from Monash's online library are clearly for undergraduate students.

Curated sources provide the highest quality of documents but at a very limited amount. This is where uncurated sources such as corporas step in. The WeeBit and Newsela corpora provide a substantial amount of documents but they are either categorized inaccurately or not categorized at all.

Therefore, python scripts were written to evaluate the overall readability spread of these data sets before we incorporate them into our final data set. We decided to use the textstat library which contains methods allowing us to average out the document readability level using multiple readability formulas. These are then recorded and aggregated to reveal the overall readability level of a dataset. Figure 5.4 below shows the resulting output of our script upon evaluating a section of the dataset - WeeBit corpora.

```
================================================
Document 187: WNL Young Americans-adv.txt
TexstatGrade: 9.0
Texstat Grade TXT: 8th and 9th grade
Category: Middle School (1)
================================================
Document 188: Zero Hours-adv.txt
TexstatGrade: 13.0
Texstat Grade TXT: 12th and 13th grade
Category: Undergraduate (3)
================================================
DISTRIBUTION
Elementary School | 0.0% (0)
Middle School     | 3.17% (6)
High School       | 25.93% (49)
Undergraduate     | 70.9% (134)
Invalid           | 0.0% (0)

OVERALL CLASSIFICATION (189 documents): 2.677248677248677
AVERAGE TEXTSTAT GRADE (189 documents): 16.555555555555557
```

(Figure 5.5: a sample distribution evaluation of documents)

From the results in Figure 5.4, we observe that over 70% of the documents are classified as Undergraduate readability with 25% being High school. Minor misclassifications into adjacent levels are to be expected, in fact it can be argued that most document's readability lies somewhere between two readability levels. Therefore we can safely categorize the documents in this dataset into the Undergraduate category of our dataset. All subsequent datasets were evaluated in a similar manner as the above to ensure an optimal distribution of datasets.

5.4 Discussion of All Results

The end product was tested and evaluated. The following sections discuss the results obtained from the project.

5.4.1 Variations of Classification Models

During the training phase of our readability classifier we used several neural network architectures and different hyperparameter configurations. Every configuration of the model was trained and documented to obtain the final readability classifier that contains the highest accuracy score.

Tester	Arch	Batch size	L2 Regulizer Factor	State Sizes	Dropout	Epochs	loss	sparse_categorical_accuracy	val_loss	val_sparse_categorical_accuracy
Alfons	bigruattention	64	0.00	128 128	0.0	25	0.2061	0.9000	0.3566	0.847
Alfons	bigruattention	64	0.00	256 256	0.0	16	0.2890	0.8671	0.4306	0.8228
Alfons	bigruattention	64	0.01	256 256	0.1	11	1.0057	0.5221	1.0186	0.5191
Alfons	bigruattention	64	0.01	256 256	0.3	5	1.0129	0.5073	1.1607	0.5073
Alfons	bigruattention	64	0.00	64 64	0.0	31	0.1668	0.9212	0.4756	0.8349
Alfons	bigruattention	64	0.00	32 32	0.0	21	0.1516	0.9293	0.467	0.8307
Alfons	bigruattention	64	0.01	128 128	0.0	30	0.7779	0.6780	0.8343	0.6483
Alfons	han	64	0.00	128 128	0.0	17	0.1572	0.9205	0.2868	0.8584
Alfons	han	64	0.00	32 32	0.0	47	0.136	0.9300	0.3591	0.87
Alfons	han	64	0.00	32 32	0.1	21	0.1211	0.9374	0.4234	0.87
Alfons	han	64	0.00	16 16	0.0	17	0.1427	0.9290	0.2989	0.8659
Alfons	han	64	0.00	16	0.0	50	0.1234	0.9393	0.308	0.8662

(Figure 5.6: records of classifier training)

From Figure 5.6 above, we focused primarily on the BiGRU attention architecture which we have discovered to yield an accuracy of about above **80%** consistently. Using other architecture such as the HAN architecture only improved the accuracy by an insignificant amount at the cost of generating a model which is twice the size of the BiGRU attention model (40MB compared to 20MB).

The state sizes in the above table refers to the number of neurons in each layer of the Gated Recurrent Unit cell. Increasing the number of state sizes will increase the size of the model significantly but contributes to a higher model accuracy.

Another common issue that we face is overfitting which tends to happen when the complexity of the model grows. In order to combat this problem, we can tweak the regularizer and dropout values during our model training.The higher the regularizer factor, the closer to 0 for the weights in the models meanwhile the higher the dropout, the more neurons will be deactivated. Validation accuracy is the most important metric we are looking at when training our model.

5.4.2 Web Application

In regards to the front end web application, it certainly achieves the outcomes we had planned during FYP1 but we felt that it is lacking the polishing touches that would make it perfect. For one, we had planned to include an extensive explanation of how the classifier works at a separate page but ended up swapping it for an embedded video explanation instead. This was because designing a documentation page to be visually appealing for common users takes a great deal of time and effort which can be invested elsewhere.

The team also feels similarly for the backend server which does satisfy all our requirements but is lacking in more advanced features. This includes security measures and support for more niche file types and additional functionalities.

5.5 Limitations of project outcomes

Limited Data Collection

Despite our best efforts to narrow down the readability categories, proper categorization of documents into the 4 categories as stated in Figure 5.2 were still not 100% accurate. Most documents we found through uncurated sources required extensive pre-processing to remove irrelevant text elements. So a lot of time had to be dedicated in cleaning and preparing documents which limited the final size of our dataset. This was partly because of the difficulty in filtering dataset obtained from undergraduate level such as research papers from Monash's online library For instance, every research paper is structured differently which makes it impossible to write an one-fits-all text processing script to trim out irrelevant document sections such as the Appendix. This in short has tremendously limited our ability to harness all the documents from the data sources we are using.

Hosting Issues
Initially we had plans to host both frontend and backend modules on cloud hosting services such as Heroku and Netlify. This would allow the teaching team to use our classifier without having to do a local manual setup for both web application and the server. We hosted the web application without any complications but were having several issues during deploying the server on Heroku. Since we had relied on Heroku's free plan, our server was limited to about 500MB of RAM which was very easily exceeded when the classification model was running. When we performed test runs on local machines, we discovered that the classification model needed about 1GB+ of RAM to process documents.

During our meetings, we discussed the possibility of AWS hosting as an alternative to Heroku as it offered more processing power. However due to time constraints, we were unable to invest the time and effort required to learn how to set up an AWS server and had to shelve the idea and settle with localhost deployment for the time being.

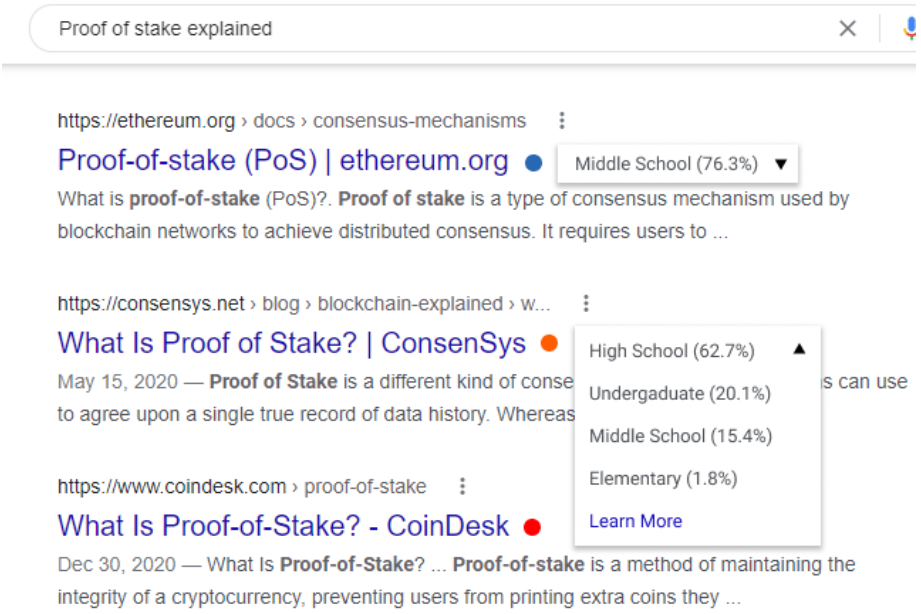
5.6 Discussion of possible improvements and future works

Any future improvements to the project will first and foremost address the shortcomings of the project outcomes. In addition we will also discuss our future ambitions should this project be hypothetically continued.

Expanding the Dataset
We believe an extensive audit to the dataset's quality and our data collection methodology is required before we start expanding the dataset. This could possibly involve using alternative means to ascertain the dataset's readability spread or enhancing existing python scripts. New sources of data would also need to be discovered and analyzed more thoroughly before they are used.

Resolving the Hosting Issue
Given the processing power needed to run the model, there are two approaches to solve this issue. The first approach is to investigate the root cause of the problem and redevelop a model which is more lightweight. However if this is not possible then the second approach would be to find alternative cloud hosting providers such as Amazon AWS or Microsoft Azure. According to a market survey, Amazon Web Services holds 33 percent of the cloud infrastructure market (Dutta, P., & Dutta, P, 2019). Investing time to learn more about AWS will not only benefit the project but also our career prospects.

Web browser Extension
The current implementation of our system requires users to visit our website and upload documents to determine their readability categories. We can make the user experience much better by implementing it as a chrome browser extension which automatically scans web pages and shows their readability classification besides the link. Below you can see what this would look like.



(Figure 5.7: UI Design based on Material Design which is used by Google for all its products)

Our planned extension would scan and extract the index.html of every web page returned by Google's search results. All these html files are sent to the backend server where text will be extracted and fed to the readability classifier. Results are then sent back and displayed to the user as color coded dots which represent the predicted readability category. Users can hover on the dots to reveal the percentage confidence of the predicted category. Clicking on the dot or on the menu will reveal the remaining percentiles as well as a link which will direct users to our web application containing all the necessary information they need to know.

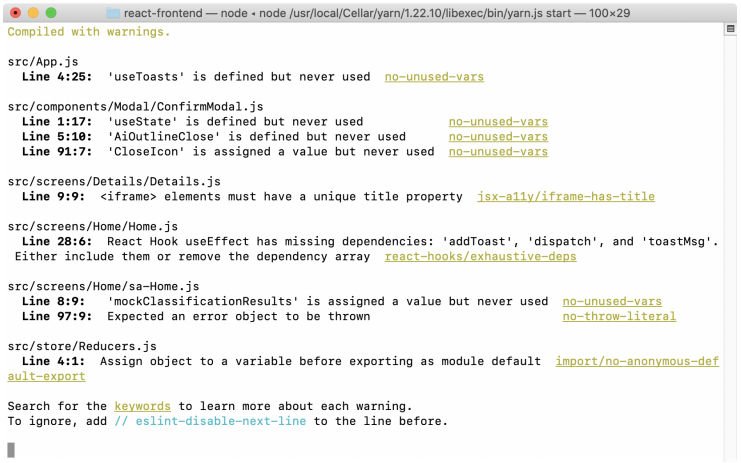
Implementing such a system would be possible given that the system architecture already supports multiple file uploads. All that is needed is to solve the hosting issue and invest time and effort in learning how to develop a chrome extension.

6 Software Deliverables

6.1 Summary of Software Deliverables

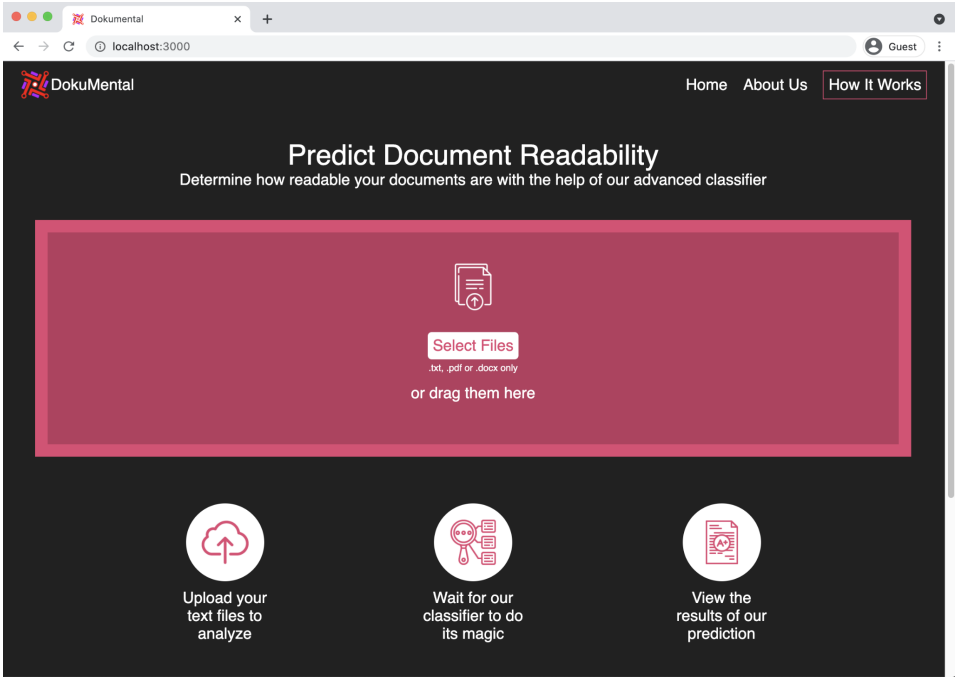
The end product of the project - a classification model is delivered with a web application that allows users to upload documents that inputs into the classification model. This web application is hosted entirely locally, where backend is hosted with Django server while frontend is displayed with a React application. To launch this application, users will be required to download both the source code for frontend and backend, and perform the setup as according to the Test Report document. A high-level description of the launching of the application is shown below.

Launching the frontend web with: `yarn start`



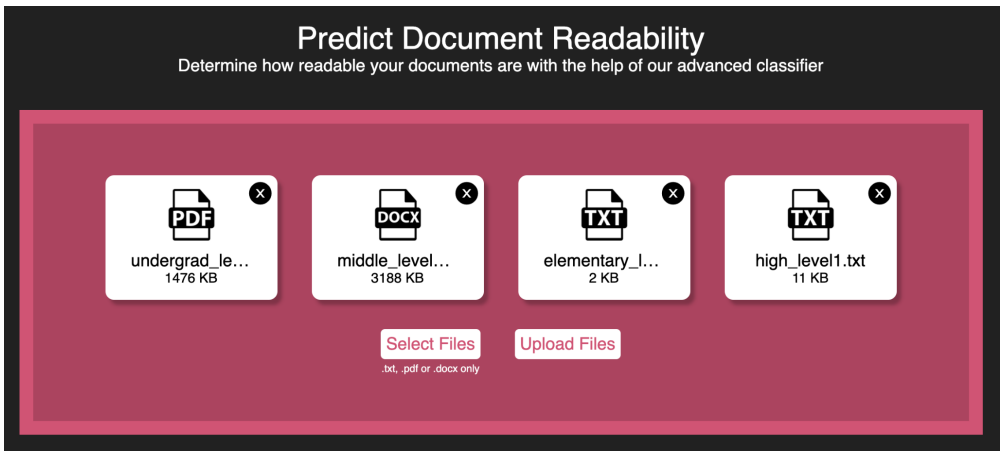
(Figure 6.1: Launching frontend)

This launches the user interface of the application on localhost:3000



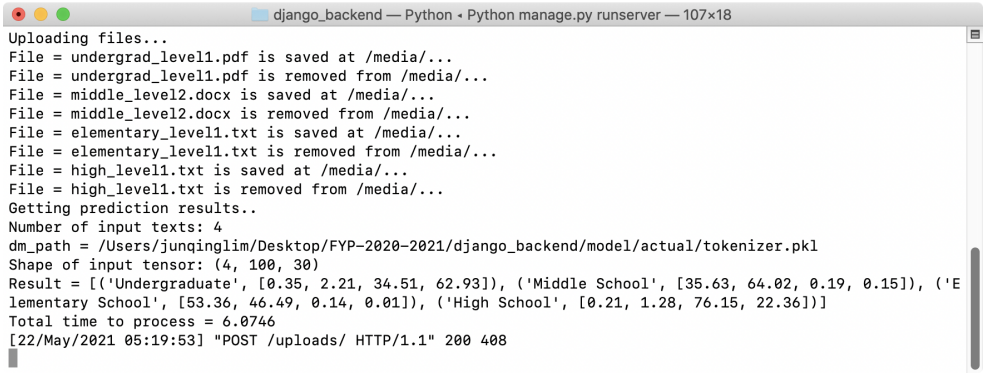
(Figure 6.2: A sample user display of the web application)

Users have two options to upload the documents, either with drag and drop or using the browser's upload manager. Document uploaded are managed by the frontend. Here is how the file dropzone would look like if we had uploaded 4 documents.



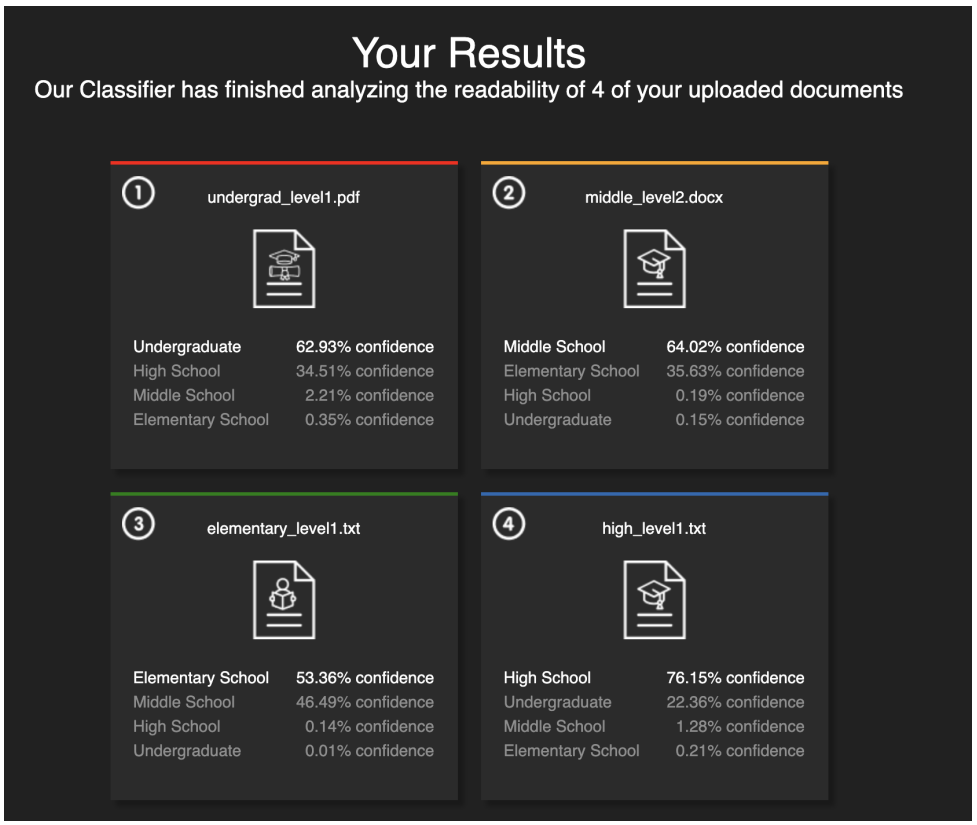
(Figure 6.3: Display of uploaded documents)

Once the documents are uploaded, the frontend will send the documents to the backend by calling the `/upload` endpoint. The backend Django server will then process the uploaded files and input them into the classification model all at once to produce the predicted readability results accordingly.



(Figure 6.4: Display of the server processing documents uploaded)

The end results of the classification model, which includes the final estimated readability level of the document and the percentages of each predicted readability level, is then returned to the frontend to be displayed to the user.



(Figure 6.5: Display of results of documents processed)

The readability level with the highest predicted percentage represents the classification model's final document category. This format of presentation makes it easy to identify "periphery" documents which the model may deem in between two readability categories. For example, in Figure 6.5, the elementary_level text document has 46.49% categorization confidence in Middle School despite being marked as elementary. This tells the user that the document may be suitable reading material for both elementary and middle school students.

6.2 Software Qualities

Various aspects of the software were considered as part of the end product deliverables. The following sections discuss the qualities of the software built.

6.2.1 Documentation and Maintainability

Documentation was mostly prioritized for the core deliverable of the project which is the implementation of the neural network architecture and the classifier training scripts. Relevant sections of the front end web application and server are documented in particular the ones involved in our test report. Furthermore the small-scale nature of the project ensures that Maintainability is preserved as both web application and server are thoroughly tested and reviewed throughout the course of the project (Plösch, Dautovic & Saft, 2014). The test report documents the software's maintainability in greater detail.

6.2.2 Security

Software security is the idea of engineering software so that it continues to function correctly under malicious attack (McGraw, 2004). Security aspect of a software plays an important role when interacting with users. Basic security measurements were implemented to protect the server:

- Only 10 documents can be uploaded at once to prevent overloading the server's CPU
- Documents uploaded can only be as large as 500MB. This prevents long execution processes by the server and prevents the launch of any adversarial attacks against the system.

As the focus of the project was on developing an optimized and improved version of a readability classification model, the backend server does not include any advanced security measures. The following scenarios can occur if the system is deployed to the public:

- Attackers uploading malware injected documents
- Coordinated flood attacks targeting the server may cause a shut down of the service (DDoS attack)
- Attackers can perform a man in the middle attack to intercept and modify request payloads between the web application and server

Any future work on expanding this project will require the development of additional security measurements, with a focus on the server security. This can range from user authentication, tracking, blacklisting suspicious IP addresses and captcha verification.

6.2.3 Robustness

The web application and server is sufficiently robust as they are capable of handling common edge cases encountered by a user. Below are some implemented features which demonstrate our system's robustness:

- Server will reject any documents which are empty (no text)
- Conversely it will also reject documents larger than 500MB to prevent overloading the CPU and model
- Server will only extract the text from documents containing a mixture of text and images
- The frontend handles all server errors by displaying either a modal pop-up or a temporary toast containing the error message

We chose to scan and handle these edge cases on the server side because of the possibility of man in the middle attacks on the HTTP payload. For example an attacker can directly use our public endpoint to send any documents to the server thus bypassing the need to use the frontend.

6.2.4 Usability

The usability of our software is the difficulty of our target users uploading and understanding the readability classification results of their documents. To achieve this, we carefully design the web application to be as user-friendly as possible.

In Figure 6.2, instructions are included below the dropzone to communicate the purpose of the application to the users. Document uploads are done through the file explorer API or just by simple drag and drop. Details of each uploaded document are clearly displayed and there is a clear button to remove unwanted documents.

The result screen has some design features that help convey readability results. Each readability level is color coded to match the following colors:

- Elementary: Green
- Middle School: Blue
- High School: Orange
- Undergraduate: Red

These colors were chosen because they are distinct and they imply each document's difficulty (green is easy and red is hard).

To measure our web application's usability, we conducted Usability testing by inviting several monash colleagues to try out our application. The test methodology, result and conclusion are documented extensively in the test report.

6.2.5 Scalability

As recommended by (Carbonara & Borrowman, 1998), a supervised machine learning model operates in the fastest mode when inputs are performed in batches. Documents uploaded are then processed by the backend and the classification model in "batches'. When receiving multiple documents in a single request, these documents are input into the classification model all at once after extracting the text from the documents. This is in contrast to the implementation of passing texts into the model for each document, which significantly reduces the time taken to process the readability predictions. This in turn provides a better overall scalability performance as the number of files increases.

Upload Type	**Filename**	**Time Taken to Classify (seconds)**	**Total Time Taken (seconds)**
A. Uploading single file for 5 times	university_50.txt	0.3776	2.0684
	university_33.txt	0.3141	
	university_34.txt	0.3389	
	university_71.txt	0.3942	
	university_80.txt	0.6436	
B. Uploading 5 files all at once	university_50.txt university_33.txt university_34.txt university_71.txt university_80.txt	1.2661	1.2661

The table above illustrates the increase in performance for an increase in number of files to be processed.

7 Critical Discussion

7.1 Comparison to Initial Project Proposal

The initial project proposal we had prepared for FYP1 focused mainly on the comparison between several neural network architectures. The literature review notably covers:

1. Bidirectional Encoder Representation from Transformers (BERT)
2. Bidirectional LSTM (BiLSTM)
3. Hierarchical Attention Network (HAN)
4. Bidirectional GRU with Attention

During the planning phase of the project, it was impossible to determine which model will work best when it comes to predicting readability levels which was why we explored a lot of approaches from different academic papers and discussed their merits.

In the end, we managed to implement three architectures namely BiLST, HAN and BiGRU Attention. All three were tested with different configurations to develop the best possible model using the final data set. The training results are shown in Section 5.4 and we ended up training our readability classifier with the Bidirectional GRU Attention architecture.

Necessary deviations such as local deployment and streamlined web application experience had to be made due to limited time constraint. Most notably, excluding them did not prevent us from accomplishing the core project requirements.

7.2 Project Execution

During the three months summer internship period, we managed to finish off the frontend web application and backend server before the semester started. Hence the team's enthusiasm during the early days of the project was quite high because we collectively believed that we could finish the project within 3 to 4 weeks time. However we soon realized that this was not the case as a lot of issues began appearing which kept pushing the deadline further and further back. Our first major hurdle was the difficulty of dataset collection. We had to spend a considerable amount of time sourcing and categorizing documents without a well defined methodology.

As we approached Week 5 we had to seek guidance from Dr Prabha in regards to the dataset issue. After a brief meeting, we were able to secure several promising sources of documents for Elementary school and Undergraduate categories. This was followed by a team meeting where we formulated a new data collection methodology and discussed key issues such as how to categorize uncurated data sources (see Section 5.3). In the meantime our technical lead was implementing the neural network architecture models we had discussed in the proposal

Data collection and model implementation formally ended on Week 7 as we had planned to start model training before Week 8. Each neural network architecture was trained multiple times under different hyperparameter configurations with each result being tabulated and recorded on a shared sheet (see Section 5.4). This allowed the Quality Assurance to rapidly perform testing while the Technical Lead analyzed and proposed new configurations to test.

After a satisfactory model was developed, the team officially entered the testing phase of our project. Our main goal was to ensure that the deliverables fulfilled all the project requirements and that they are of high quality. To start off the testing phase, the Project Lead took one of the early generated models and began his work on implementing it onto the backend server. This in turn revealed several bugs which were quickly reported to the technical lead so necessary changes could be made and tested by the Quality Assurance.

There were issues with hosting the server online as we had discussed so we had to resort to local deployment. Each module owner performed unit tests before Integration and System testing. The team also invited our colleagues from Monash to perform third party Usability testing to observe how real users interacted with our system. All the test results are compiled in our Test report where we analyze and discuss our findings more thoroughly.

7.3 Reflection on Outcomes

Now that we have reached the end of our project, we can say that it was a success as we had managed to develop and flesh out all the planned requirements in the initial proposal. Our interest in the topic of readability gave us a lot of ideas to expand our project further and potentially develop a minimum viable product that can be released for public use (see Section 5.6).

As IT professionals, this project gave us the opportunity to cultivate our skills in technical fields such as neural networks, front end development and back end development. We also dabbled in topics such as cloud hosting for system deployment and data scraping to automate data collection to achieve our system requirements. Moving forward, we believe that working on projects which go beyond our comfort zone in terms of our understanding of technologies would be greatly beneficial for personal development.

As a team, we started the project with a naive mindset and almost no sense of planning. We were always too eager to do tasks without properly formulating a solid methodology and plan. This led to several avoidable issues notably the data set delay we had discussed previously. By closely adhering to the Waterfall project management methodology and strictly following the project's Gantt chart, we were able to work more efficiently for the latter half of the semester. Thus we as a team come to learn the importance of proper team management and are able to take the lessons learnt in this project and hopefully apply them in our upcoming careers to set a good example.

8 Conclusion

In conclusion, our project has successfully developed readability classifiers using machine learning architectures of Bidirectional GRU with Attention Neural Networks (BiGRU) and Hierarchical Attention Network (HAN). These classifiers fully utilized machine learning techniques such as regularization, grid search, model fine-tuning to obtain an optimal performance. We then performed testing using various hyperparameter configurations to search for the best model. The final model we ended up choosing was developed using the BiGRU Attention network with a readability classification accuracy of 87%. The classifier is pickled and embedded into the back-end server hosted in real-time which will communicate with a web application to evaluate all submitted document readability. Readability prediction results in the form of confidence percentiles are displayed to the user thus fulfilling the requirements of the project.

The team also performed 5 stages of testing for each module, ensuring that any foreseeable edge cases were handled correctly. Immediate future project improvements were also identified to resolve current system limitations. Our expansion plan provides a strong foundation to develop a minimum viable product if one decides to continue working on this project. In the future we hope that this readability classifier will be able to be embedded directly or indirectly within a user's retrieval system (such as a Chrome extension, or word document, etc.). This would improve the classification process of retrieval systems which will take into account the readability of documents alongside relevancy thus resulting in greater user satisfaction.

9 References

Verma, M., Yilmaz, E., & Craswell, N. (2016, February). On obtaining effort based judgements for information retrieval. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining (pp. 277-286).

Danet, B., Herring, S. C., & Herring, S. C. (Eds.). (2018). The multilingual Internet: Language, culture, and communication online. Oxford University Press on Demand.

Elizabeth, H, 3 Types of Essential Resources For Your Project, 18 May 2016, Retrieved from: pmtips.net/blog-new/3-types-of-essential-resources-for-your-project.

Dutta, P., & Dutta, P. (2019). Comparative Study of Cloud Services Offered by Amazon, Microsoft & Google. International Journal of Trend in Scientific Research and Development, 3(3), 981-985.

Martinc, M., Pollak, S., & Robnik-Šikonja, M. (2019). Supervised and unsupervised neural approaches to text readability. *arXiv preprint arXiv:1907.11779*.

Davison, A., & Kantor, R. N. (1982). On the failure of readability formulas to define readable texts: A case study from adaptations. *Reading research quarterly*, 187-209.

Crossley, S. A., Skalicky, S., Dascalu, M., McNamara, D. S., & Kyle, K. (2017). Predicting text comprehension, processing, and familiarity in adult readers: New approaches to readability formulas. *Discourse Processes*, *54*(5-6), 340-359.

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems* (pp. 649-657).

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., & Černocký, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth annual conference of the international speech communication association*.

Sun, Y., Chen, K., Sun, L., & Hu, C. (2020, July). Attention-based Deep Learning Model for Text Readability Evaluation. *In 2020 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE.

Mohammadi, H., & Khasteh, S. H. (2019). Text as Environment: A Deep Reinforcement Learning Text Readability Assessment Model. arXiv preprint arXiv:1912.05957.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016, June). Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies* (pp. 1480-1489).

Caspers, M. (2017). React and Redux. Rich Internet Applications w/HTML and Javascript, 11.

Pantula, M., & Kuppusamy, K. S. (2020). A Machine Learning-Based Model to Evaluate Readability and Assess Grade Level for the Web Pages. *The Computer Journal.*

Krasner, G. E., & Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3), 26-49.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1), 63-90.

Tensorflow. (2021). Customization basics: tensors and operations. Retrieved from https://www.tensorflow.org/tutorials/customization/basics

Robby Neale. (2019). Introducing TF.Text. Retrieved from https://blog.tensorflow.org/2019/06/introducing-tftext.html

Bonaccorso, G. (2018). Mastering machine learning algorithms: expert techniques to implement popular machine learning algorithms and fine-tune your models. *Packt Publishing Ltd.*

Meyer, K. (2015). Toward a Definition of Minimalism: Principles of Minimal Visual Design in Web Interfaces.

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *arXiv preprint arXiv:1205.6904.*

Raz, T., Shenhar, A. J., & Dvir, D. (2002). Risk management, project success, and technological uncertainty. *R&d Management*, 32(2), 101-109.

Norman, D. A., & Nielsen, J. (2010). Gestural interfaces: a step backward in usability. interactions, 17(5), 46-49.

Euphemia Wong. (2020). Shneiderman's Eight Golden Rules Will Help You Design Better Interfaces. Retrieved from https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces

Plösch, R., Dautovic, A., & Saft, M. (2014, October). The value of software documentation quality. *In 2014 14th International Conference on Quality Software* (pp. 333-342). IEEE.

McGraw, G. (2004). Software security. *IEEE Security & Privacy*, 2(2), 80-83.

Carbonara, L., & Borrowman, A. (1998, September). A comparison of batch and incremental supervised learning algorithms. *In European Symposium on Principles of Data Mining and Knowledge Discovery* (pp. 264-272). Springer, Berlin, Heidelberg.

Dataset Sources

EReading Worksheets
https://www.ereadingworksheets.com/free-reading-worksheets/reading-comprehension-worksheets/

A Levels English Literature Past papers
https://pastpapers.papacambridge.com/papers/caie/cambridge-advanced-as-and-a-level-english-literature-9695

IELTS Reading
https://www.ielts-exam.net/ielts_reading/

ISL Collective
https://en.islcollective.com/english-esl-worksheets/search?material-type=reading-comprehension-activities&level=elementary-a1

OneStop
https://github.com/nishkalavallabhi/OneStopEnglishCorpus

WeeBit
https://www.aclweb.org/anthology/W12-2019.pdf

10 Numbered Annex

Appendix A: Source Codes

Data Preprocessing Source Code

```
"""
All readability scores will be recategorized into the following:
  Elementary school 0 (5th grade and under)
  Middle school     1 (6th grade to 9th grade)
  High school       2 (10th grade to 12th grade)
  Undergraduate     3 (Undergraduate and up)
"""
def normalize_grade(grade):
    if grade < 6:
        return 0
    elif 6 <= grade < 10:
        return 1
    elif 10 <= grade < 13:
        return 2
    elif grade >= 13:
        return 3
    else:
        return -1

def category_name(category):
    if category == 0:
        return "Elementary"
    elif category == 1:
        return "Middle School"
    elif category == 2:
        return "High School"
    elif category == 3:
        return "Undergraduate"
    else:
        return ""

def text_valid(txt=""):
    return txt.strip() != ""

# Reset report
with open("!READABILITY_REPORT.txt", "w", encoding="utf8") as fw:
    fw.write("=============== READABILITY REPORT ===============\n")

txt_files = glob.glob("*.txt")
if "!READABILITY_REPORT.txt" in txt_files:
    txt_files.remove("!READABILITY_REPORT.txt")  # exclude the report

tally = [0, 0, 0, 0, 0]
total_sum = 0
textat_grade_sum = 0
```

```
for i, txt_file in enumerate(txt_files):
    with open(txt_file, "r", encoding="utf8") as fd:
        try:
            extracted_text = fd.read()
            # Check if text is valid (not empty)
            if text_valid(extracted_text):
                textstat_grade = textstat.text_standard(
                    extracted_text, float_output=True)
                textstat_grade_text = textstat.text_standard(
                    extracted_text, float_output=False)

                normalized_grade = normalize_grade(textstat_grade)
            else:
                textstat_grade = textstat_grade_text = "N/A"
                normalized_grade = -1
            # Categorize the result (only add to total_sum if valid)
            tally[normalized_grade] += 1
            if normalized_grade != -1:
                total_sum += normalized_grade
                textat_grade_sum += textstat_grade
            # Write to report and console
              report_txt = f"""Document {i}: {txt_file} \nTexstatGrade: {textstat_grade}
\nTexstat Grade TXT: {textstat_grade_text} \nCategory: {category_name(normalized_grade)}
({normalized_grade})\n{SEPARATOR}"""
            with open("!READABILITY_REPORT.txt", "a", encoding="utf8") as fw:
                fw.write(report_txt)
            print(report_txt)
        except:
            print("Failed to calculate document readability")

elementary_percent = round(tally[0] / sum(tally) * 100, 2)
middle_percent = round(tally[1] / sum(tally) * 100, 2)
high_percent = round(tally[2] / sum(tally) * 100, 2)
university_percent = round(tally[3] / sum(tally) * 100, 2)
invalid_percent = round(tally[4] / sum(tally) * 100, 2)

valid_documents = sum(tally[:4])
overall_score = total_sum / valid_documents  # exclude invalids from total
textstat_grade_mean = textat_grade_sum / valid_documents

final_report = f"""DISTRIBUTION
Elementary School | {elementary_percent}% ({tally[0]})
Middle School     | {middle_percent}% ({tally[1]})
High School       | {high_percent}% ({tally[2]})
University School | {university_percent}% ({tally[3]})
Invalid           | {invalid_percent}% ({tally[4]})

OVERALL CLASSIFICATION ({valid_documents} documents): {overall_score}
AVERAGE TEXTSTAT GRADE ({valid_documents} documents): {textstat_grade_mean}
"""
# Write & print final report
with open("!READABILITY_REPORT.txt", "a", encoding="utf8") as fw:
    fw.write(final_report)
print(final_report)
```

Classification Model (DataManager) Source Code

Note: The code below is a minor extraction of the implementation of classification model, for a deeper inspection on the code, please refer to the Code Upload (in separate submission)

```
class DataManager:
         def   __init__(self,   path,   train_ratio=0.8,   verbose=False,   vocab_size=5000,
encoding='utf8'):
        nltk.download('punkt')
        if len(path) <= 0:
            raise AttributeError('Invalid dataset path.')
        self.path = path
        self.verbose = verbose
        self.train_ratio = train_ratio
        self.vocab_size = vocab_size
        self.encoding = encoding
        self.texts = []
        self.sentences = []
        self.labels = []
        self.load_text_files()
        self.encode_labels()
        self.tokenize()
        self.train_valid_split()
        with open("tf_tokenizer.pkl", "wb") as pkl_file:
            pickle.dump(self.tokenizer, pkl_file)

    def split_into_sentences(self, text):
        alphabets= "([A-Za-z])"
        prefixes = "(Mr|St|Mrs|Ms|Dr)[.]"
        suffixes = "(Inc|Ltd|Jr|Sr|Co)"
                                                                    starters         =
"(Mr|Mrs|Ms|Dr|He\s|She\s|It\s|They\s|Their\s|Our\s|We\s|But\s|However\s|That\s|This\s|W
herever)"
        acronyms = "([A-Z][.][A-Z][.](?:[A-Z][.])?)"
        websites = "[.](com|net|org|io|gov)"

        text = " " + text + "  "
        text = text.replace("\n"," ")
        text = re.sub(prefixes,"\\1<prd>",text)
        text = re.sub(websites,"<prd>\\1",text)
        if "Ph.D" in text: text = text.replace("Ph.D.","Ph<prd>D<prd>")
        text = re.sub("\s" + alphabets + "[.] "," \\1<prd> ",text)
        text = re.sub(acronyms+" "+starters,"\\1<stop> \\2",text)
               text  =  re.sub(alphabets  +  "[.]"  +  alphabets  +  "[.]"  +  alphabets  +
"[.]","\\1<prd>\\2<prd>\\3<prd>",text)
        text = re.sub(alphabets + "[.]" + alphabets + "[.]","\\1<prd>\\2<prd>",text)
        text = re.sub(" "+suffixes+"[.] "+starters," \\1<stop> \\2",text)
        text = re.sub(" "+suffixes+"[.]"," \\1<prd>",text)
        text = re.sub(" " + alphabets + "[.]"," \\1<prd>",text)
        if """ in text: text = text.replace(".""",""".")
        if "\"" in text: text = text.replace(".\"","\".")
        if "!" in text: text = text.replace("!\"","\"!")
        if "?" in text: text = text.replace("?\"","\"?")
        text = text.replace(".",".<stop>")
        text = text.replace("?","?<stop>")
        text = text.replace("!","!<stop>")
        text = text.replace("<prd>",".")
```

```
        sentences = text.split("<stop>")
        sentences = sentences[:-1]
        sentences = [s.strip() for s in sentences]
        return sentences

    def load_text_files(self):
        tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
        print("Dataset path:", self.path)
        for classes in glob.glob(os.path.join(self.path, "*")):
            for text in glob.glob(os.path.join(classes, "*")):
                self.labels.append(os.path.basename(classes))
                text = open(text, encoding=self.encoding).read().lower()
                self.texts.append(text)
                text = tokenizer.tokenize(text)
                self.sentences.append(text[: 100])

    def encode_labels(self):
        le = preprocessing.LabelEncoder()
        self.class_id = le.fit_transform(self.labels)
        self.labels = le.classes_
        self.num_classes = len(self.labels)
                self.max_sentence_number  =  max(map(lambda  sentence:  len(sentence),
self.sentences))
        self.max_sentence_number = 100

    def tokenize(self):
        self.tokenizer = Tokenizer(self.vocab_size)
        self.tokenizer.fit_on_texts(self.texts)
           self.tokens  =  [self.tokenizer.texts_to_sequences(self.sentences[i])  for  i  in
range(len(self.sentences))]
            max_array  =  [list(map(lambda  sentence:  len(sentence),  text))  for  text  in
self.tokens]
        self.maxlen = max(max(max_array))
        self.maxlen = 30
         self.tokens = [pad_sequences(self.tokens[i], padding='post', truncating='post',
value=0, maxlen=self.maxlen) for i in range(len(self.tokens))]
        for i in range(len(self.tokens)):
            if self.tokens[i].shape[0] < self.max_sentence_number:
                for _ in range(self.max_sentence_number - self.tokens[i].shape[0]):
                      self.tokens[i] = np.append(self.tokens[i], [np.zeros(self.maxlen)],
0)
        self.word2idx = self.tokenizer.word_index
        self.vocab_size = len(self.word2idx)

    def train_valid_split(self):
        idxs = np.random.permutation(len(self.tokens))
        train_size = int(self.train_ratio * len(idxs)) + 1
        self.tokens = np.asarray(self.tokens)[idxs]
        self.class_id = np.asarray(self.class_id)[idxs]
                    self.train_tokens,  self.valid_tokens  =  self.tokens[:train_size],
self.tokens[train_size:]
                  self.train_classes,  self.valid_classes  =  self.class_id[:train_size],
self.class_id[train_size:]
        self.train_tokens = np.asarray(self.train_tokens)
        self.valid_tokens = np.asarray(self.valid_tokens)
        self.train_set = Dataset.from_tensor_slices((train_tokens, train_classes))
```

```
        self.val_set = Dataset.from_tensor_slices((valid_tokens, valid_classes))
```

Web App - Frontend Source Code (React - File Input Validations)

```
const DropZone = ({ toggleErrorModal, toggleConfirmModal }) => {
    const dispatch = useDispatch();
    const submittedFiles = useSelector((state) => state.Home.submittedFiles);

    const onDrop = useCallback(
        (inputFiles) => {
        const checkFiles = (inFiles) => {
            let exists = false;
            inFiles.forEach((x) => {
              submittedFiles.forEach((y) => {
                if (x.path === y.path) exists = true;
              });
        });
        return exists;
      };
      if (submittedFiles.length + inputFiles.length > 10) {
        toggleErrorModal("You can only upload a maximum of 10 documents");
      } else if (checkFiles(inputFiles)) {
        toggleErrorModal("File already uploaded");
      } else {
        dispatch(
          actHome.handleState("submittedFiles", [
            ...submittedFiles,
            ...inputFiles,
          ])
        );
      }
    },
    [dispatch, submittedFiles, toggleErrorModal]
  );

  const onDropRejected = () =>
        toggleErrorModal("Only txt, docx and pdf files are allowed");

  const { getRootProps, getInputProps, isDragActive, open } = useDropzone({
        onDrop,
        onDropRejected,
        accept: ".docx, .txt, .pdf",
        noClick: true,
  });

  const cardTransition = useTransition(submittedFiles, (file) => file.path, {
        from: { opacity: 0, transform: "scale(0)" },
        enter: { opacity: 1, transform: "scale(1)" },
        leave: { opacity: 0, transform: "scale(0)" },
        config: { mass: 1, tension: 300, friction: 18 },
  });

  const fileIcon = (type) => {
```

```
  switch(type){
      case "text/plain": return txt_icon;
      case "application/pdf": return pdf_icon;
      default: return docx_icon;
  }
}

const renderDocuments = () => {
  return cardTransition.map(({ item, key, props }) => {
      const { name, type, path, size } = item;
      return (
      <DocCard key={key} style={props}>
        <CloseIcon
          onClick={() =>
            dispatch(
              actHome.handleState(
                "submittedFiles",
                submittedFiles.filter((item) => item.path !== path)
              )
            )
          }
        >
          <AiFillCloseCircle />
        </CloseIcon>
        <img
          className="icon"
          src={fileIcon(type)}
          alt=".docx"
        />
        <div>
          <h1>{name}</h1>
          <p>{Math.round(size / 1024, 2)} KB</p>
        </div>
      </DocCard>
    );
  });
};

return (
  <OutterWrapper {...getRootProps({})}>
      <input {...getInputProps({})} />
      <InnerWrapper isDragActive={isDragActive}>
      {submittedFiles.length === 0 && (
        <img className="icon" src={UploadImg} alt="Upload Documents Here" />
      )}
      <DocWrapper>{renderDocuments()}</DocWrapper>
      <ButtonGroup>
        <div>
          <SelectButton onClick={open}>Select Files</SelectButton>
          <p style={{ marginTop: "4px", fontSize: "12px" }}>
            .txt, .pdf or .docx only
          </p>
        </div>
        {submittedFiles.length !== 0 && (
          <div>
            <SelectButton
```

```
                onClick={() =>
                  toggleConfirmModal(
                    "Are you sure you want to submit these documents?"
                  )
                }
              >
                Upload Files
              </SelectButton>
            </div>
          )}
        </ButtonGroup>
        {submittedFiles.length === 0 && (
          <p>{!isDragActive ? "or drag them here" : "and drop them"}</p>
        )}
        </InnerWrapper>
    </OutterWrapper>
  );
};
```

Web App - Backend Source Code (Django - File Upload)

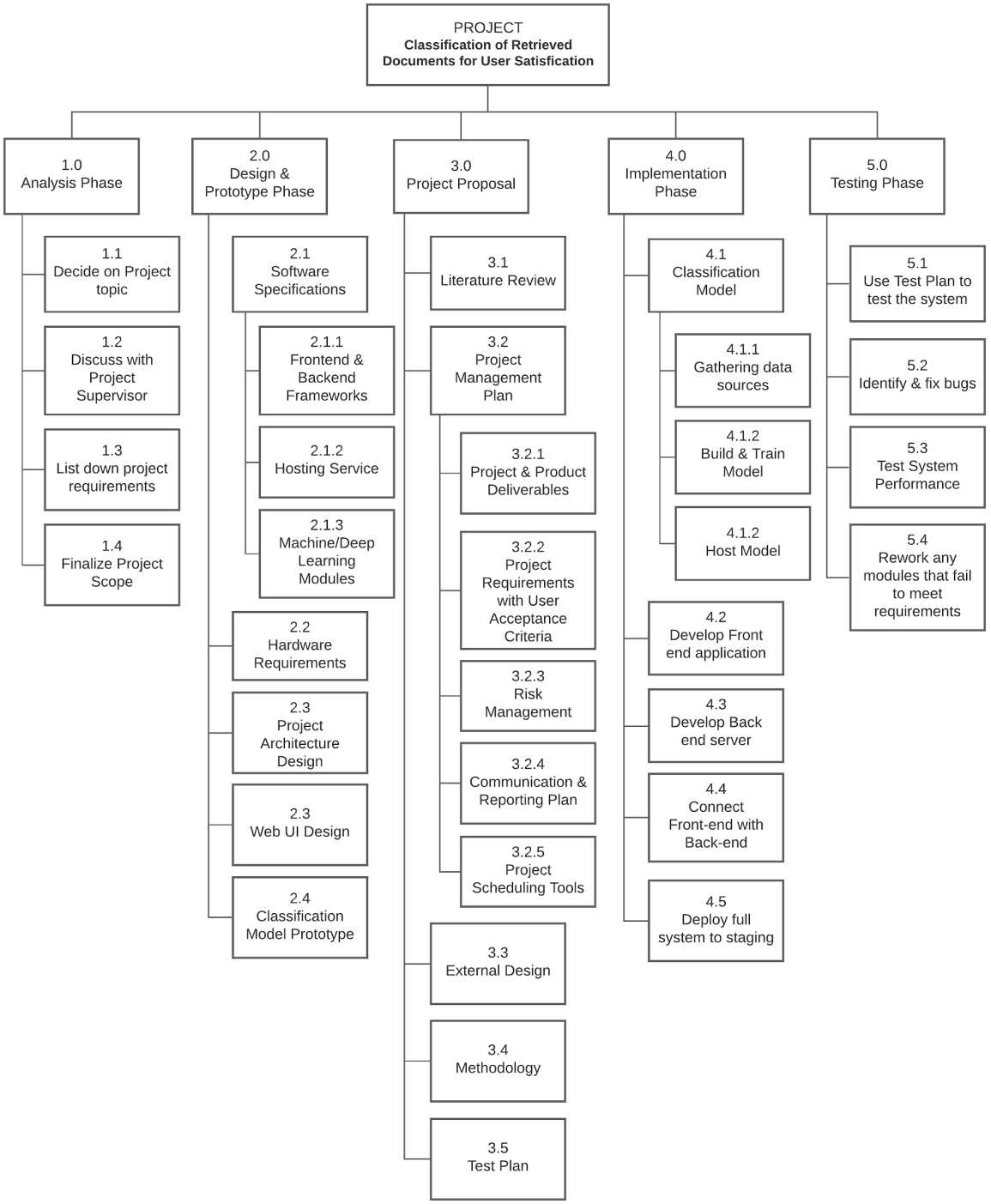
```
def extract_texts(files):
    """Function to extract text from a list of files"""
    arr = []
    for file in files:
        fs = FileSystemStorage()
        file_uploaded = fs.save(file.name, file)
        file_path = fs.location + "/" + file_uploaded
        print(f'File = {file} is saved at {fs.base_url}...')

        text = textract.process(file_path)
        text = text.decode('utf-8')
        arr.append(text)

        fs.delete(file_path)
        print(f'File = {file} is removed from {fs.base_url}...')
    return arr

@csrf_exempt
def upload_files(request):
    """Function to upload files using system file uploader"""
    print("Uploading files...")
    if request.method == 'POST':
        form = UserUploadForm(request.POST, request.FILES)
        files = request.FILES.getlist('file')

        if form.is_valid():
            # Wrong file indication
            if len(files) == 0:
                    return HttpResponse("Please indicate upload file field as 'file'",
status=400)
            start_time = timer()
            # Extract the text from the files
            arr_of_texts = extract_texts(files)
```

```
            # Run the model to get the result, in the format of a list of tuple (level,
percentages)
            results = run_model(arr_of_texts)
            print(f'Result = {results}')
            end_time = timer()

            print(f'Total time to process = {round(end_time - start_time, 4)}')
            # Parse into return JSON format
            upload_results = []
            for i in range(len(results)):
                if len(arr_of_texts[i]) > 0:
                    temp_res = {
                        "name": files[i].name,
                        "level": results[i][0],
                        "percentages": results[i][1]
                    }
                    upload_results.append(temp_res)
                else:
                    temp_res = {
                        "name": files[i].name,
                        "level": "Fail",
                        "description": "File is empty"
                    }
                    upload_results.append(temp_res)

            return JsonResponse(upload_results, safe=False)

    return Response("Unable to upload files", status=status.HTTP_403_FORBIDDEN)
```

Appendix B: Final Report Workload Distribution

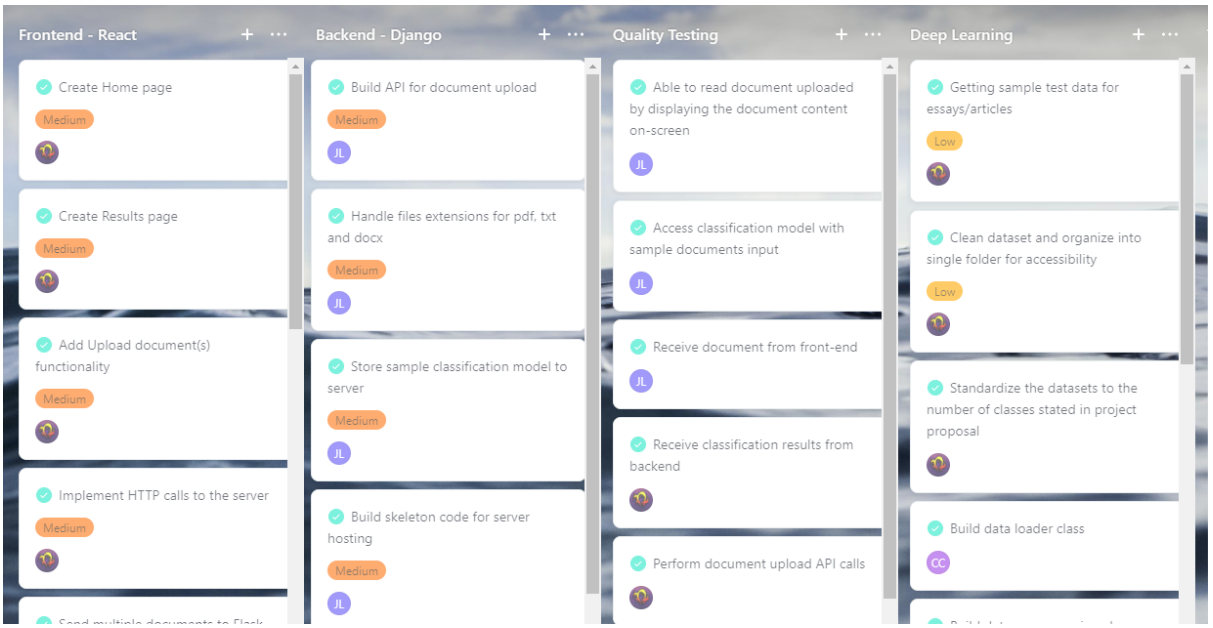
No	Section	Team Members
1	Introduction	● Alfons Fernaldy
2	Background	● Chang Yin Cheng ● Lim Jun Qing
3	Methodology	● Chang Yin Cheng ● Lim Jun Qing
4	Project Management	● Alfons Fernaldy
5	Outcomes	● Alfons Fernaldy ● Lim Jun Qing
6	Software Deliverables	● Chang Yin Cheng ● Lim Jun Qing
7	Critical Discussion	● Alfons Fernaldy
8	Conclusion	● Chang Yin Cheng

Appendix C: Work Breakdown Agreement



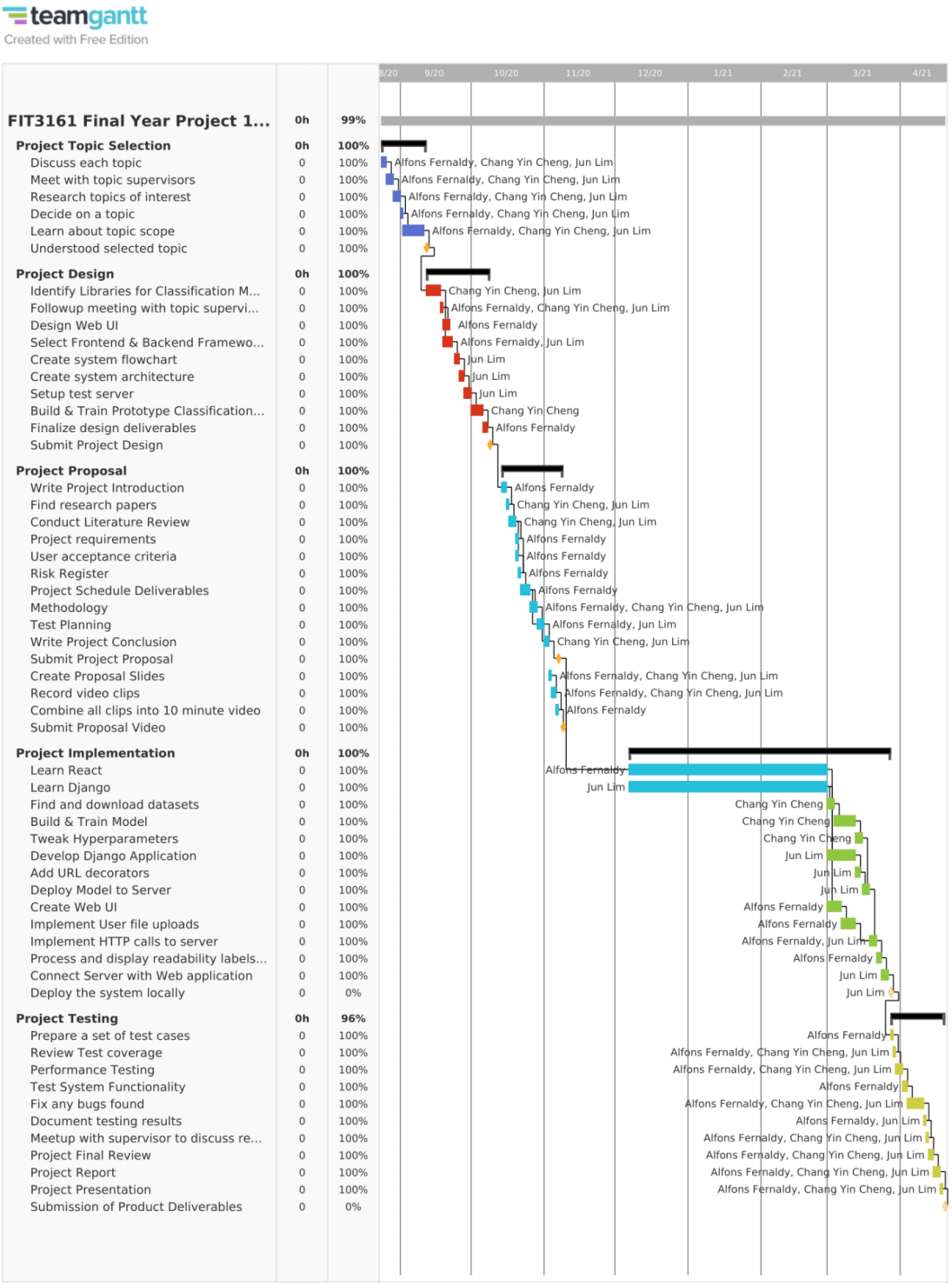
Appendix D: Kanban Board

Completed Kanban board to manage tasks between system modules.



Appendix E: Gantt Chart

Completed Gantt Chart project timeline.



Appendix F: Risk Register

ID	Rank	Description	Category	Root Cause	Triggers	Potential Responses	Risk Owner	Probability	Impact	Risk	Status	Risk Rationale
1	5	Poor Data Diversity	Technical	Data sources used to train the classification model lack diversity in terms of expected reader's age.	Insufficient and/or poorly selected data sources used to train model	Seek more diverse datasets which cover reading materials for all ages and educational backgrounds	Project Manager, Software Developer	3	6	18	Open	This risk is well know within the team as it has caused our early prototype classifier models to poorly classify documents of other age groups. Therefore it is unlikely that we will overlook this when building the real model.
2	6	Server Incompatibility with model and web application	Technical	Backend server may be incompatible when it comes to hosting the model or communicating with the front end module.	Failure to host the model and/or receive HTTP request calls from the web application.	Look for libraries or external modules that would resolve any incompatibility issues. If all else fails, switch to an alternative as detailed during project design.	Project Team	2	8	16	Open	During testing, we had successfully hosted our prototype model using the server therefore we anticipate that this will hold true in the future. If such a scenario arises then we may have to drastically change our server infrastructure.
3	7	Web application cannot handle/upload user submitted documents	Technical	Front end web application is unable to take in and/or send documents to the backend server.	Failure to implement file upload during the development of the web application.	Look for libraries or solutions within the extensive React community.	Software Developer, Quality Assurance	2	6	12	Open	When deciding our frontend web framework, we chose React over its alternatives due to its huge developer community. Hence we expect the availability of libraries and support which allow us to implement features such as file uploading.
4	4	Slow document processing	Technical	Classification model struggles to classify submitted documents quickly which leads to performance bottlenecks	Heavy server traffic or large documents.	Analyse and tweak the model to improve its performance without sacrificing predictive accuracy.	Project Team	4	5	20	Open	The final performance of the model is unknown but it is a risk that should be anticipated especially when the model is deployed live for users to use. A slow service will negatively impact user experience.
5	1	Insufficient time to learn necessary skills.	Schedule	Tight schedules and other units coupled with the burden of internships can leave almost no time to learn the necessary skills needed for the project	Insufficient learning time allocated to learn development/project skills.	Come up with a schedule and roadmap to ensure necessary skills are learnt before the project officially starts.	Project Manager	6	6	36	Open	Each team member schedules time differently and it is difficult to predict available time during the internship period of each member. Failure to properly learn necessary skills can slow down the project as members will have to take time to learn.
6	2	Addition or Modification of Project Requirements	Scope	A core requirement needs to be modified/added to fulfill the project scope.	Consultation from the project supervisor or internal discussions.	Hold a meeting and readjust project schedules to take into account of any changes.	Project Team	5	5	25	Open	Project Requirements are subject to change if either the team or the project supervisor feel that it is necessary. Obviously it is hard to predict the scale of changes and whether they will heavily detract the team from the original schedule.
7	8	Unavailable Supervisor Support	Schedule	The project supervisor is unable to allocate time to discuss the project with us.	Supervisor is busy with handling other units or teams.	Discuss alternate means of communication and support.	Project Manager	4	3	12	Open	Our supervisor has so far been able to make time to answer our questions even before committing to the topic however this can change. We also expect to have a good understanding of the project scope early on and would only need to meet up with our supervisor for necessary audits.
8	3	Monash Campus remains closed for the foreseeable future	External	The growing Covid-19 cases forces Monash to not open its doors for yet another semester.	Sustained spread of the coronavirus especially within Malaysia and Indonesia.	Discuss contingency plans to manage project development remotely.	Project Team	5	4	20	Open	It is in the team's best interest to be able to meet face to face especially when the project kicks off. This makes it easy to share ideas and collaborate in creating each module. Unfortunately we cannot know for sure if such a privilege can be afforded given the risks.