



SRI RAMACHANDRA

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Category - I Deemed to be University) Porur, Chennai

SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

Cyberbully Detection and Analytics On Online Media

INTERNSHIP REPORT

Quarter IV (Year 1)

Submitted by

MAGESH BOOPATHI

E0121040

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Cyber Security & Internet of Things)

Sri Ramachandra Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai -600116

JULY, 2020



SRI RAMACHANDRA

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Category - I Deemed to be University) Porur, Chennai

SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that this project report “**Cyberbully Detection and Analytics On Online Media**” is the bonafide work of **MAGESH BOOPATHI Reg No. E0121040** who carried out the internship work under my supervision.

Signature of Faculty Mentor

Signature of Vice-Principal

Dr. Jayanthi Ganapathy

Prof. M. Prema

Assistant Professor

Vice-Principal

Sri Ramachandra Engineering and Technology

Sri Ramachandra Engineering and Technology

Porur

Porur

Chennai-600116

Chennai-600116

Evaluation Date:

Title	Page
1. Domain Introduction	5
1.1 Cyberbullying	5
1.2 Literature Review and Analysis	5
1.3 Harmful effects	6
2. Objective	8
3. Work Flow and Architecture	9
3.1 Data Mining	9
3.1.1 Wikipedia Corpus	9
3.1.2 Book Corpus	11
3.1.3 Building the Data-set	13
3.2 Building the Model	15
3.2.1 Data preparation	15
3.2.2 Implementation	18
3.2.3 Classifier	22
3.3 API Integration	23
4. Tools and Technology Used	24
4.1 Git and GitHub	24
4.2 PyTorch	24
4.3 Amazon EC2	24
4.4 Amazon Lake Formation	25
5. Results and Output	26
5.1 Data analysis on Cyberbullying	26
5.2 Model training results	27
5.3 API Integration Example with Discord.js	28
6. Timeline	30
7. Scope for further enhancement	31
Conclusion	32
References	33

ACKNOWLEDGEMENT

I express my sincere gratitude to our Chancellor, Vice-Chancellor and our sincere gratitude to our Provost **Dr. Raju** and our Vice-Principal **Prof. Prema** for their support and for providing the required facilities for carrying out this study.

I wish to thank my faculty mentor, **Prof. Jayanthi Ganapathy** Department of Computer Science and Engineering, Sri Ramachandra Engineering and Technology for extending help and encouragement throughout the project. Without his/her continuous guidance and persistent help, this project would not have been a success for me.

I am grateful to Department of Computer Science and Engineering, Sri Ramachandra Engineering and Technology, our beloved parents and friends for extending the support, who helped us to overcome obstacles in the study.

Images

1. Figure 1 - Survey on Cyberbullying
2. Figure 2 - Data Mining Flow
3. Figure 3 - Wiki HTML Files
4. Figure 4 - Converted Text Files
5. Figure 5 - JSON Data from Google API
6. Figure 6 - Book Crawler Running
7. Figure 7 - Books Converted To Text Files
8. Figure 8 - S3 Data Lake
9. Figure 9 - Data Stored in Lake
10. Figure 10 - Loading the Custom Data-set
11. Figure 11 - 10,000 vocabulary built using WordPiece
12. Figure 12 - Preprocessed Data
13. Figure 13 - Attention Architecture
14. Figure 14 - Transformer Model
15. Figure 15 - BERT Training
16. Figure 16 - BERT Classifier Training
17. Figure 17 - Most Used Hate words
18. Figure 18 - Forms of Cyberbullying
19. Figure 19 - Classifier Accuracy
20. Figure 20 - Classifier Loss
21. Figure 21 - Language Model Loss
22. Figure 22 - Deleting messages and warning users
23. Figure 23 - Logging Cyberbullying related events
24. Figure 24 - Timeline June 29 - July 16
25. Figure 25 - Timeline July 10 - July 29

1. Domain Introduction

1.1 Cyberbullying

In simple words, cyberbullying is bullying with the use of digital technologies. It takes place often on digital platforms like social media, messaging platforms, gaming platforms and multiplayer games. It involves offensive behaviour intended to shame, scare or anger a person. For example :

- Sending hate speech, hurtful, abusive or threatening messages, images, videos and other media on social platforms.
- Impersonating someone and sending mean messages to others on their behalf or through fake accounts.
- Spreading lies about or posting embarrassing photos or videos of someone on social media.

Unlike face-to-face bullying, cyberbullying leaves a digital footprint and can be used as evidence to help combat the issue.

1.2 Literature Review and Analysis

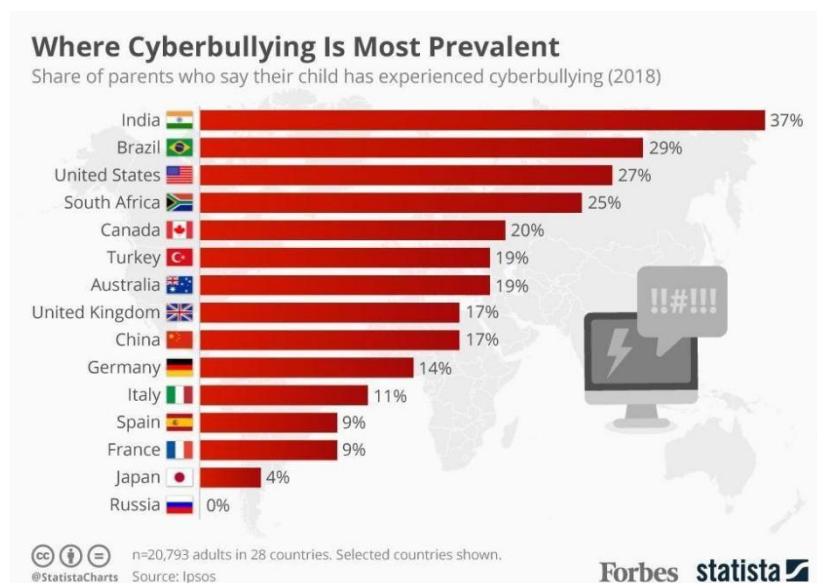


Figure 1 - Survey on Cyberbullying

As shown in Fig. 1, a survey was conducted on 20,000 parents worldwide about high-risk online platforms, “65% single out cyberbullying on social media as their biggest fear. Other common threats include text messaging (38%) and chat rooms (34%).” Clearly India seems to be most affected by cyberbullying, with factors such as high population and easy access to internet in recent times.

1.3 Harmful effects

It's no surprise that cyberbullying, like regular face-to-face bullying can be detrimental to the mental health of a person. Experiences of cyberbullying can cause someone to feel ashamed, nervous and insecure about the opinions of others. It can lead to withdrawal from social interactions and make them feel subdued.

It could destroy self-confidence and motivation to do the things are usually enjoyable and create feelings of isolation from the people you love and trust. For young people, skipping school is another common effect of cyberbullying and they may turn to substances like alcohol and drugs or violent behaviour to deal their psychological and physical pain. The effects of cyberbullying on mental health can vary depending on the medium through which it happens.

2. Objective

1. Data Mining and Preparation
2. Model Training and Optimization
3. Model inference through a HTTP API

Making use of the digital evidence which cyberbullying leaves behind, we can combat this issue perhaps at the very early stages. With the tech and tools we have today, it's possible to make use of machine learning and artificial intelligence to detect hate speech and offensive content in a text message.

Hate speech detection through machine learning can be useful for content moderation in different social media platforms. The objective is to build a machine learning model which takes text as parameters and classifies the content as offensive, hateful or normal through a scale of 0 to 1. This classification can be useful to implement auto moderation in social platforms such as warning the users or blocking the messages from being sent.

To provide a single solution to all platforms, an API such a simple HTTP API can be could be used to communicate with language model for receiving model classification on the input text. An HTTP is easy to integrate in social and gaming platform to check cyberbullying.

3. Work Flow and Architecture

3.1 Data Mining

To train the language model, a data-set containing text should be created through mining it from the required sources. This could be accomplished in 2 different ways :

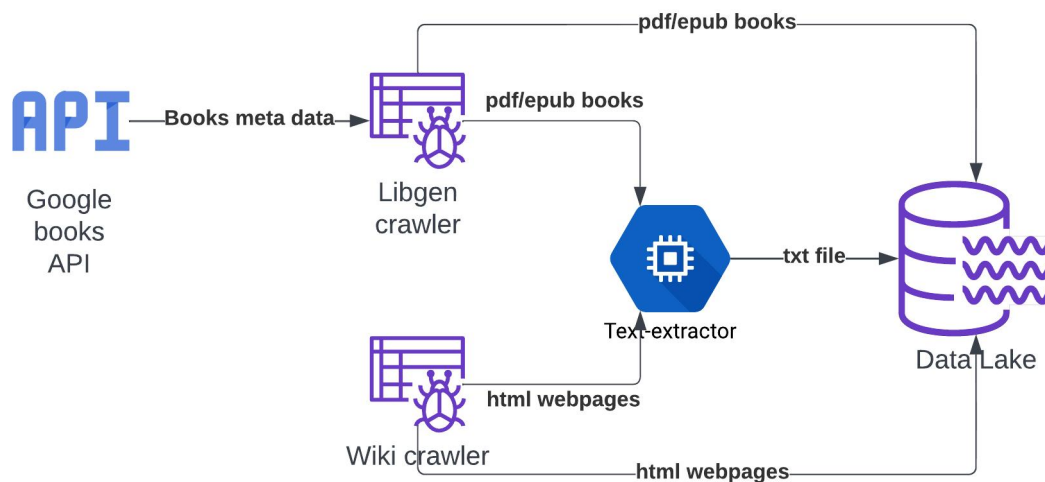


Figure 2 - Data Mining Flow

3.1.1 Wikipedia Corpus

Wikipedia is a non-profit organization which provides articles which contain reference links and hosts other projects. It is a reliable source for data mining, hence a web crawler was built in python to crawl through Wikipedia. The web-pages are queried using the requests module and the HTML content are scraped using the BeautifulSoup module as shown in Fig. 3 and continues crawling and collecting text data in Wikipedia. The collected text for each web-page article is saved to their respective .txt files.

Name	Date Modified	Size	Kind
wiki-Alisa_Garland.html	04-Jul-2022 at 1:35 AM	48 KB	HTML document
wiki-Alexa_Internet.html	04-Jul-2022 at 1:35 AM	147 KB	HTML document
wiki-Alison_Settle.html	04-Jul-2022 at 1:35 AM	46 KB	HTML document
wiki-Andree_Clark_Bird_Refuge.html	04-Jul-2022 at 1:35 AM	56 KB	HTML document
wiki-Audrey-Withers.html	04-Jul-2022 at 1:35 AM	65 KB	HTML document
wiki-Bagulo.html	04-Jul-2022 at 1:35 AM	484 KB	HTML document
wiki-Beatrix_Miller.html	04-Jul-2022 at 1:35 AM	56 KB	HTML document
wiki-Bella_Hadid.html	04-Jul-2022 at 1:35 AM	370 KB	HTML document
wiki-Bulgari.html	04-Jul-2022 at 1:35 AM	238 KB	HTML document
wiki-Christ_Church_Nazareth.html	04-Jul-2022 at 1:35 AM	53 KB	HTML document
wiki-Cz%C4%99stochowa.html	04-Jul-2022 at 1:35 AM	408 KB	HTML document
wiki-Dorothy_Todd.html	04-Jul-2022 at 1:35 AM	53 KB	HTML document
wiki-East_Beach_(Santa_Barbara).html	04-Jul-2022 at 1:35 AM	43 KB	HTML document
wiki-Edmonde_Charles-Roux.html	04-Jul-2022 at 1:35 AM	120 KB	HTML document
wiki-Edna_Woolman_Chase.html	04-Jul-2022 at 1:35 AM	57 KB	HTML document
wiki-Elspeth_Champcommunal.html	04-Jul-2022 at 1:35 AM	64 KB	HTML document
wiki-F%C3%A9lix_Bonfils.html	04-Jul-2022 at 1:35 AM	106 KB	HTML document
wiki-File/Bella_Hadid_Cannes_2018_2.jpg.html	04-Jul-2022 at 1:35 AM	70 KB	HTML document
wiki-File/Create-protection-shackle.svg.html	04-Jul-2022 at 1:35 AM	56 KB	HTML document
wiki-File/Move-protection-shackle.svg.html	04-Jul-2022 at 1:35 AM	61 KB	HTML document
wiki-File/Nazareth.3women.jpg.html	04-Jul-2022 at 1:35 AM	42 KB	HTML document
wiki-File/Pending-protection-shackle.svg.html	04-Jul-2022 at 1:35 AM	54 KB	HTML document
wiki-File/Semi-protection-shackle.svg.html	04-Jul-2022 at 1:35 AM	55 KB	HTML document
wiki-File/Template-protection-shackle.svg.html	04-Jul-2022 at 1:35 AM	57 KB	HTML document
wiki-File/Upload-protection-shackle.svg.html	04-Jul-2022 at 1:35 AM	56 KB	HTML document
wiki-Fran%C3%A7oise_de_Langlade.html	04-Jul-2022 at 1:35 AM	37 KB	HTML document
wiki-Galliee.html	04-Jul-2022 at 1:35 AM	176 KB	HTML document
wiki-Gigi_Hadid.html	04-Jul-2022 at 1:35 AM	319 KB	HTML document
wiki-H&M.html	04-Jul-2022 at 1:35 AM	314 KB	HTML document
wiki-IMG_Models.html	04-Jul-2022 at 1:35 AM	78 KB	HTML document
wiki-Ines_G_%C5%BDupanov.html	04-Jul-2022 at 1:35 AM	61 KB	HTML document
wiki-John_Quigley_(academic).html	04-Jul-2022 at 1:35 AM	57 KB	HTML document
wiki-JSTOR_(identifier).html	04-Jul-2022 at 1:35 AM	152 KB	HTML document
wiki-Main_Bocher.html	04-Jul-2022 at 1:35 AM	55 KB	HTML document
wiki-Met_Gala.html	04-Jul-2022 at 1:35 AM	311 KB	HTML document
wiki-Mohamed_Hadid.html	04-Jul-2022 at 1:35 AM	112 KB	HTML document
wiki-Nazareth.html	04-Jul-2022 at 1:35 AM	407 KB	HTML document
wiki-New_York_Fashion_Week.html	04-Jul-2022 at 1:35 AM	155 KB	HTML document
wiki-Poland.html	04-Jul-2022 at 1:35 AM	1.1 MB	HTML document
wiki-Prabal_Gurung.html	04-Jul-2022 at 1:35 AM	68 KB	HTML document
wiki-Richard_T._Antoun.html	04-Jul-2022 at 1:35 AM	95 KB	HTML document

Figure 3 - Wiki HTML Files

Name	Date Modified	Size	Kind
Alisa_Garland__Wikipedia.txt	04-Jul-2022 at 1:35 AM	2 KB	Plain Text
Alexa_Internet__Wikipedia.txt	04-Jul-2022 at 1:35 AM	9 KB	Plain Text
Alison_Settle__Wikipedia.txt	04-Jul-2022 at 1:35 AM	2 KB	Plain Text
Andree_Clark_Bird_Refuge__Wikipedia.txt	04-Jul-2022 at 1:35 AM	3 KB	Plain Text
Audrey-Withers__Wikipedia.txt	04-Jul-2022 at 1:35 AM	6 KB	Plain Text
Bagulo__Wikipedia.txt	04-Jul-2022 at 1:35 AM	38 KB	Plain Text
Beatrix_Miller__Wikipedia.txt	04-Jul-2022 at 1:35 AM	3 KB	Plain Text
Bella_Hadid__Wikipedia.txt	04-Jul-2022 at 1:35 AM	37 KB	Plain Text
Bulgari__Wikipedia.txt	04-Jul-2022 at 1:35 AM	13 KB	Plain Text
Christ_Church_Nazareth__Wikipedia.txt	04-Jul-2022 at 1:35 AM	2 KB	Plain Text
Cz%C4%99stochowa__Wikipedia.txt	04-Jul-2022 at 1:35 AM	36 KB	Plain Text
Dorothy_Todd__Wikipedia.txt	04-Jul-2022 at 1:35 AM	4 KB	Plain Text
East_Beach_(Santa_Barbara)__Wikipedia.txt	04-Jul-2022 at 1:35 AM	953 bytes	Plain Text
Edmonde_CharlesRoux__Wikipedia.txt	04-Jul-2022 at 1:35 AM	3 KB	Plain Text
Edna_Woolman_Chase__Wikipedia.txt	04-Jul-2022 at 1:35 AM	2 KB	Plain Text
Elspeth_Champcommunal__Wikipedia.txt	04-Jul-2022 at 1:35 AM	4 KB	Plain Text
F%C3%A9lix_Bonfils__Wikipedia.txt	04-Jul-2022 at 1:35 AM	4 KB	Plain Text
File/Bella_Hadid_Cannes_2018_2.jpg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	1 KB	Plain Text
File/Createprotectionshackle.svg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	726 bytes	Plain Text
File/Moveprotectionshackle.svg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	871 bytes	Plain Text
File/Nazareth.3women.jpg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	1 KB	Plain Text
File/Pendingprotectionshackle.svg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	810 bytes	Plain Text
File/Semiprotectionshackle.svg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	807 bytes	Plain Text
File/Templateprotectionshackle.svg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	875 bytes	Plain Text
File/Uploadprotectionshackle.svg__Wikipedia.txt	04-Jul-2022 at 1:35 AM	708 bytes	Plain Text
Fran%C3%A7oise_de_Langlade__Wikipedia.txt	04-Jul-2022 at 1:35 AM	1 KB	Plain Text
Galliee__Wikipedia.txt	04-Jul-2022 at 1:35 AM	23 KB	Plain Text
Gigi_Hadid__Wikipedia.txt	04-Jul-2022 at 1:35 AM	14 KB	Plain Text
H&M__Wikipedia.txt	04-Jul-2022 at 1:35 AM	21 KB	Plain Text
IMG_Models__Wikipedia.txt	04-Jul-2022 at 1:35 AM	1 KB	Plain Text
Ines_G_%C5%BDupanov__Wikipedia.txt	04-Jul-2022 at 1:35 AM	4 KB	Plain Text
John_B_Quigley__Wikipedia.txt	04-Jul-2022 at 1:35 AM	1 KB	Plain Text
JSTOR__Wikipedia.txt	04-Jul-2022 at 1:35 AM	11 KB	Plain Text
Main_Bocher__Wikipedia.txt	04-Jul-2022 at 1:35 AM	2 KB	Plain Text
Met_Gala__Wikipedia.txt	04-Jul-2022 at 1:35 AM	14 KB	Plain Text
Mohamed_Hadid__Wikipedia.txt	04-Jul-2022 at 1:35 AM	6 KB	Plain Text
Nazareth__Wikipedia.txt	04-Jul-2022 at 1:35 AM	46 KB	Plain Text
New_York_Fashion_Week__Wikipedia.txt	04-Jul-2022 at 1:35 AM	7 KB	Plain Text
Poland__Wikipedia.txt	04-Jul-2022 at 1:35 AM	78 KB	Plain Text
Prabal_Gurung__Wikipedia.txt	04-Jul-2022 at 1:35 AM	4 KB	Plain Text
Richard_T._Antoun__Wikipedia.txt	04-Jul-2022 at 1:35 AM	9 KB	Plain Text

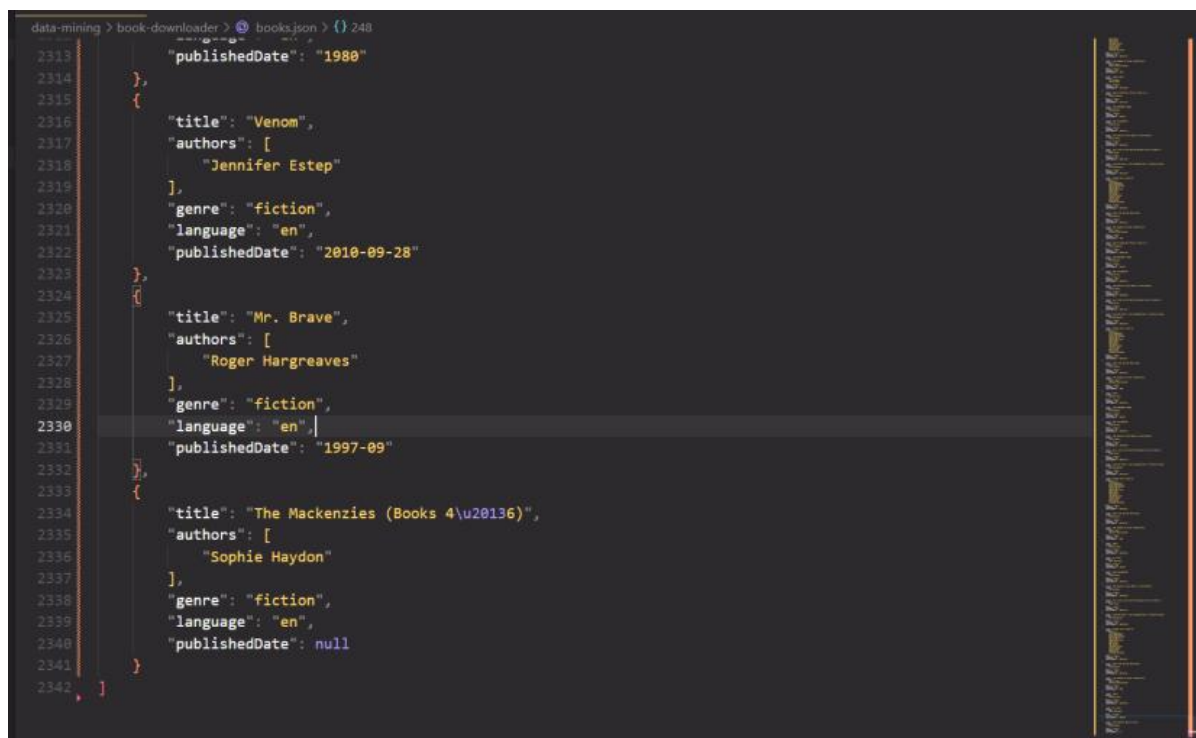
Figure 4 - Converted Text Files

This way, a crawler was used to mine text data from Wikipedia, and proves to be a reliable source for building a data-sets.

3.1.2 Book Corpus

From the perspective of data mining, digital books are huge chunks of text which are an excellent source of data. This proves to be a reliable source for building the data-set as well.

However the method to retrieve text data from books is not very straight forward. It requires querying a list of books from an API such as Google Books API. Google Books API provides a feature to search books by various metadata such as name, genre, author and so on. For the purpose of mining data, 5 genres were selected, and a book list was constructed by querying the API. The collected metadata about books were used to retrieve the digital books through a python crawler.



```
data-mining > book-downloader > books.json > {} 248
2313     "publishedDate": "1980"
2314   },
2315   {
2316     "title": "Venom",
2317     "authors": [
2318       "Jennifer Estep"
2319     ],
2320     "genre": "fiction",
2321     "language": "en",
2322     "publishedDate": "2010-09-28"
2323   },
2324   {
2325     "title": "Mr. Brave",
2326     "authors": [
2327       "Roger Hargreaves"
2328     ],
2329     "genre": "fiction",
2330     "language": "en",
2331     "publishedDate": "1997-09"
2332   },
2333   {
2334     "title": "The Mackenzies (Books 4\u20136)",
2335     "authors": [
2336       "Sophie Haydon"
2337     ],
2338     "genre": "fiction",
2339     "language": "en",
2340     "publishedDate": null
2341   }
2342 ]
```

Figure 5 - JSON Data from Google API

Library Genesis is an online book library which provides books in various formats. Using the metadata that was collected, a python crawler was used to search the website for each book. The crawler was built to collect download links of .epub and .pdf formats were collected and downloaded, as shown in Fig. 6.

```

PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
Python + + + + +

Finished Parsing HTML response
Download Page Link : https://libgen.rocks/ads.php?md5=3419bef31584a5baf69f85cd9e272c24
Book Download Link : https://libgen.rocks/get.php?md5=3419bef31584a5baf69f85cd9e272c24&key=IPT4FL53RQ5I004B
Downloading Book ...
Saving Book ...
Jim Collins, Jerry I. Porras - Built to last_ successful habits of visionary companies (2002, HarperBusiness Essentials) - libgen.lc.epub
Finished Downloading Book!

Search Query = The+Subtle+Art+of+Not+Giving+a+f*ck+Mark+Manson
Finished Parsing HTML response
Download Page Link : https://libgen.rocks/ads.php?md5=5f831906c063f29b2ff13a4026112bc1
Book Download Link : https://libgen.rocks/get.php?md5=5f831906c063f29b2ff13a4026112bc1&key=MBBLJ3EGHC6Z4QKY
Downloading Book ...
Saving Book ...
Mark Manson, - The Subtle Art of Not Giving a F_ck_ A Counterintuitive Approach to Living a Good Life (2016, Harper) - libgen.li.pdf
Finished Downloading Book!

Search Query = Ikigai+Hector+Garcia
Finished Parsing HTML response
Download Page Link : https://libgen.rocks/ads.php?md5=1c0657844e26557f189f5543ac1ddc81
Book Download Link : https://libgen.rocks/get.php?md5=1c0657844e26557f189f5543ac1ddc81&key=1TLV6D1ZYIQR6KXX
Downloading Book ...
Saving Book ...
Hector Garcia - Ikigai_ The Japanese Secret to a Long and Happy Life (2017, Penguin Books) - libgen.lc.pdf
Finished Downloading Book!

Search Query = The+Kite+Runner+Khaled+Hosseini
Finished Parsing HTML response
Download Page Link : https://libgen.rocks/ads.php?md5=f672d38d537a165e87c9a0723d12d849
Book Download Link : https://libgen.rocks/get.php?md5=f672d38d537a165e87c9a0723d12d849&key=WH0JQ53MWF14I30
Downloading Book ...
Saving Book ...
Khaled, Hosseini - The Kite runner (2009) - libgen.li.epub
Finished Downloading Book!

```

Figure 6 - Book Crawler Running

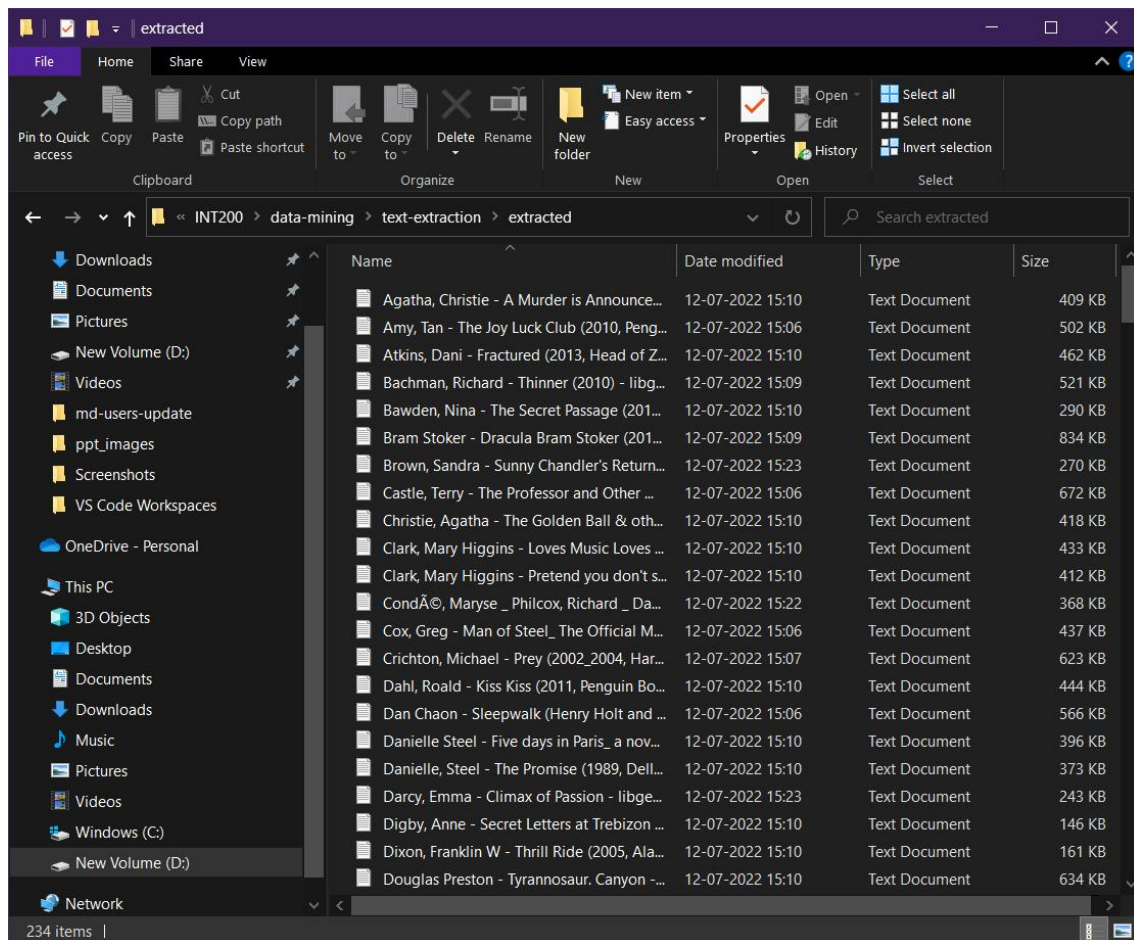


Figure 7 - Books Converted To Text Files

One last step remains before the downloads books can be useful for building the data-set. The text has to be extracted from each book into .txt files just like for the wiki articles. ebooklib is useful to extract text content from .epub books. Similarly, the PyPDF2 python module can be used to extract from a .pdf file.

3.1.3 Building the Data-set

Since its not practical to download and save the text data locally, it's a better approach to directly upload the content to AWS Data-Lake for storage purposes. The Data-Lake will be the data-set for training the Language Model.

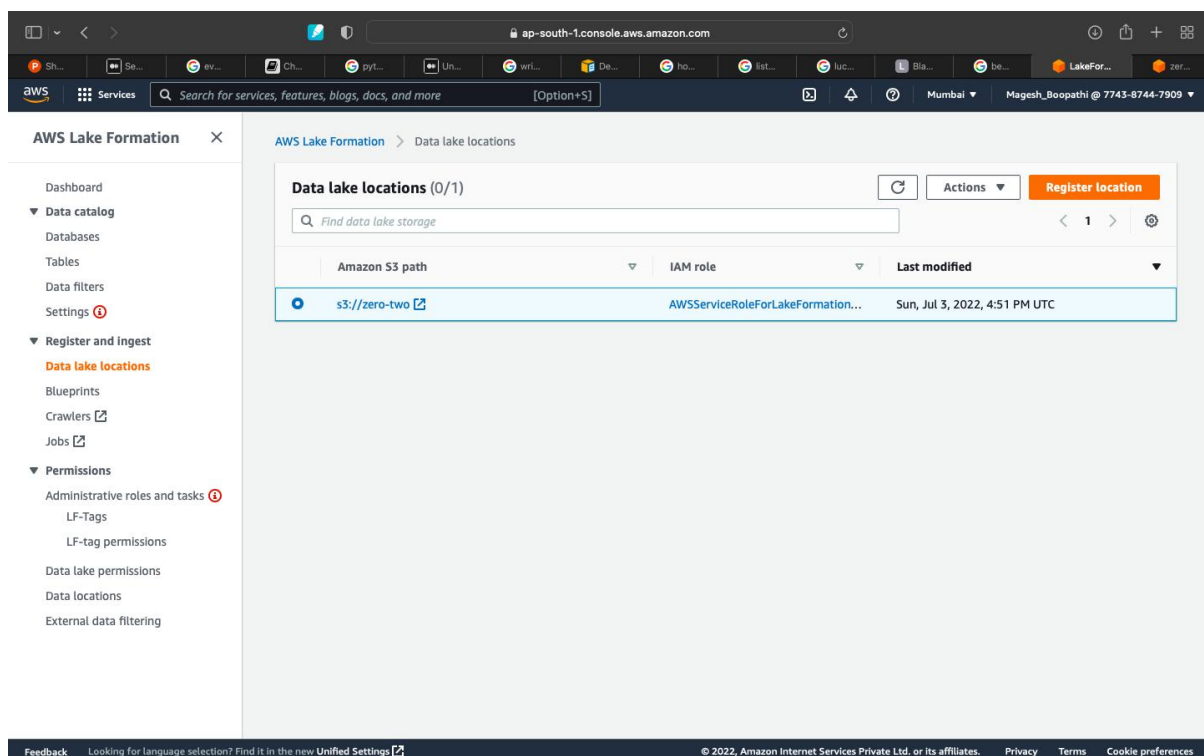


Figure 8 - S3 Data Lake

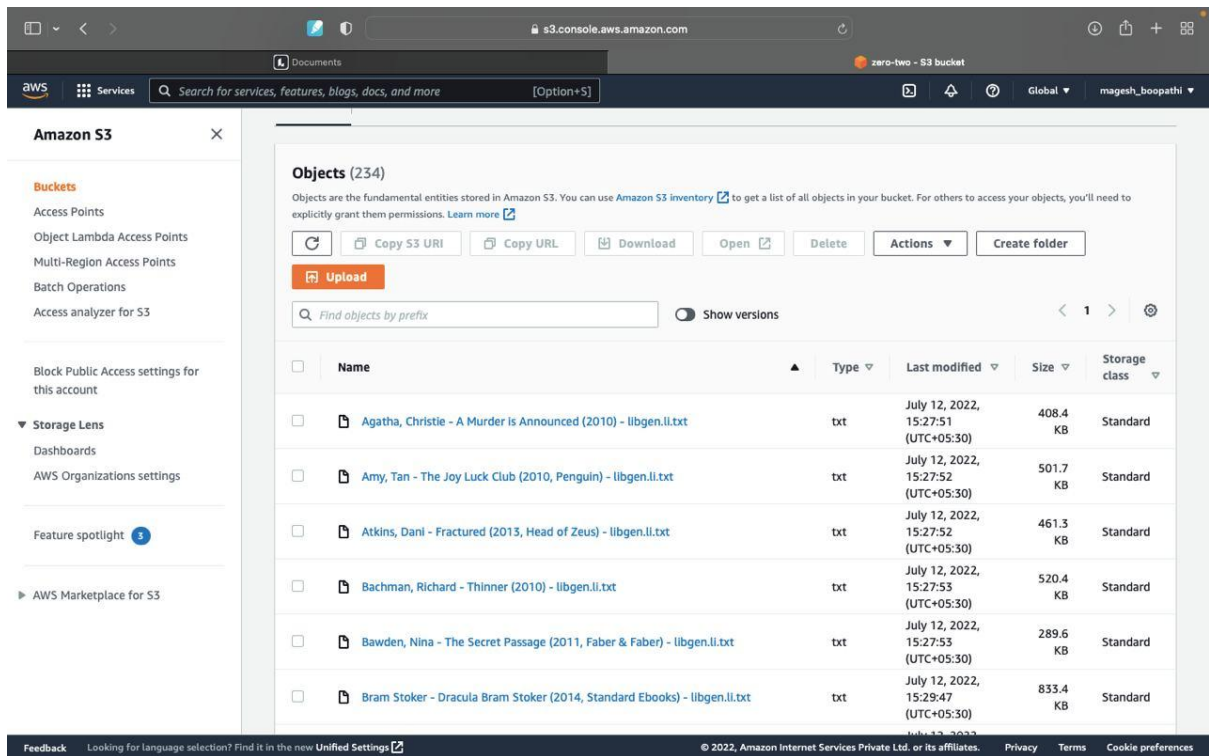


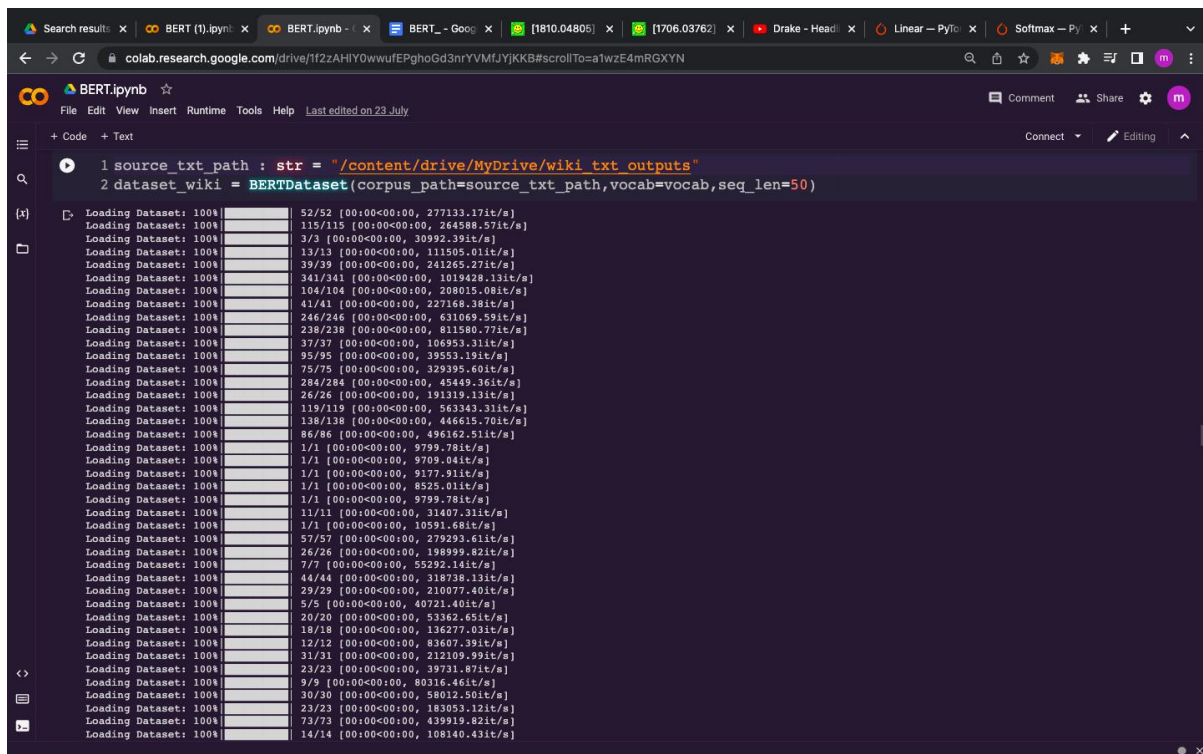
Figure 9 - Data Stored in Lake

3.3 Building the model

3.3.1 Data preparation

In order to pre-train any language model, a large amount of words from a particular language is required. BERT was built using Books Corpus (800M words) and English Wikipedia (2500M words). [1]

In order to obtain similar result, we built our own book corpus and English Wikipedia corpus.



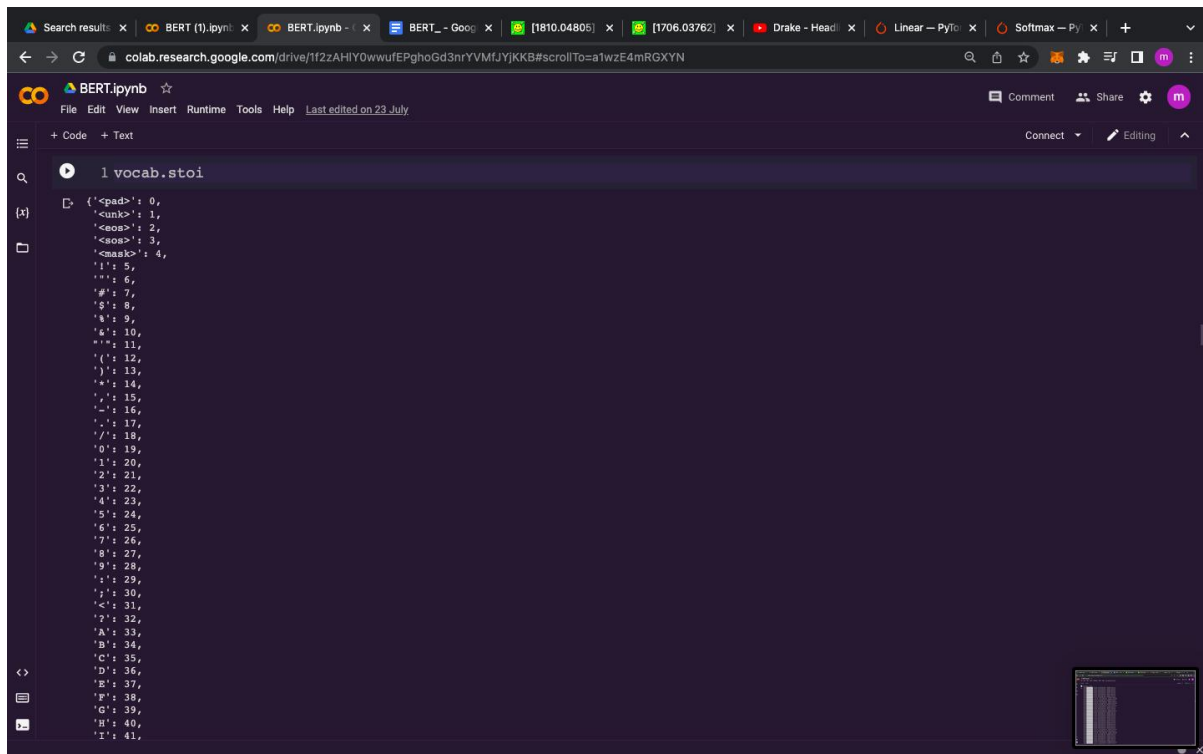
```
1 source_txt_path : str = "/content/drive/MyDrive/wiki txt outputs"
2 dataset_wiki = BERTDataset(corpus_path=source_txt_path,vocab=vocab,seq_len=50)
```

File	Size	Speed
52/52	00:00:00:00	277133.17it/s
115/115	00:00:00:00	264588.57it/s
3/3	00:00:00:00	30992.39it/s
13/13	00:00:00:00	111505.01it/s
39/39	00:00:00:00	241265.27it/s
341/341	00:00:00:00	1019428.13it/s
104/104	00:00:00:00	208015.08it/s
47/47	00:00:00:00	227168.38it/s
246/246	00:00:00:00	631069.59it/s
238/238	00:00:00:00	811580.77it/s
37/37	00:00:00:00	106953.31it/s
95/95	00:00:00:00	39553.19it/s
75/75	00:00:00:00	329395.60it/s
284/284	00:00:00:00	49449.36it/s
26/26	00:00:00:00	191319.13it/s
119/119	00:00:00:00	563343.31it/s
138/138	00:00:00:00	446615.70it/s
86/86	00:00:00:00	496162.51it/s
1/1	00:00:00:00	9799.78it/s
1/1	00:00:00:00	9709.04it/s
1/1	00:00:00:00	9177.51it/s
1/1	00:00:00:00	8525.01it/s
1/1	00:00:00:00	9799.78it/s
11/11	00:00:00:00	31407.31it/s
1/1	00:00:00:00	10591.68it/s
57/57	00:00:00:00	279293.61it/s
26/26	00:00:00:00	198999.82it/s
7/7	00:00:00:00	55292.14it/s
44/44	00:00:00:00	318738.13it/s
29/29	00:00:00:00	210077.40it/s
5/5	00:00:00:00	40721.40it/s
20/20	00:00:00:00	53362.65it/s
18/18	00:00:00:00	136277.03it/s
12/12	00:00:00:00	83607.39it/s
31/31	00:00:00:00	212109.99it/s
23/23	00:00:00:00	39731.87it/s
9/9	00:00:00:00	80316.46it/s
30/30	00:00:00:00	58012.50it/s
23/23	00:00:00:00	183053.12it/s
73/73	00:00:00:00	439919.82it/s
14/14	00:00:00:00	108140.43it/s

Figure 10 - Loading the Custom Data-set

To make BERT handle a variety of down-stream tasks, input representation proposed in the paper is able to unambiguously represent both a single sentence and a pair of sentences in one token sequence.

Original BERT uses WordPiece embedding with a 30K token vocabulary. Whereas due to technical limitations we used WordPiece embedding with 10K token vocabulary.



```
1 vocab.stoi
{
  '<pad>': 0,
  '<unk>': 1,
  '<eos>': 2,
  '<eos>': 3,
  '<mask>': 4,
  '0': 5,
  '1': 6,
  '2': 7,
  '3': 8,
  '4': 9,
  '5': 10,
  '6': 11,
  '7': 12,
  '8': 13,
  '9': 14,
  ' ': 15,
  ' ': 16,
  ' ': 17,
  ' ': 18,
  '0': 19,
  '1': 20,
  '2': 21,
  '3': 22,
  '4': 23,
  '5': 24,
  '6': 25,
  '7': 26,
  '8': 27,
  '9': 28,
  ' ': 29,
  ' ': 30,
  '<': 31,
  '>': 32,
  'A': 33,
  'B': 34,
  'C': 35,
  'D': 36,
  'E': 37,
  'F': 38,
  'G': 39,
  'H': 40,
  'I': 41,
}
```

Figure 11 - 10,000 vocabulary built using WordPiece

The first token of every sequence is always a special classification token ([CLS]). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.

The sentences are differentiated in two ways. First, we separate them with a special token ([SEP]). Second, we add a learned embedding to every token indicating whether it belongs to sentence A or sentence B.

BERT is trained unsupervised on the following two tasks :

1. Masked token prediction (MLM) : 15% of the input tokens are masked randomly and the model is trained to predict the original tokens. This helps the LM to reach deep bidirectional representation.
2. Next token prediction (NSP) : Many NLP tasks are based on understanding the relationship between two sentences. When choosing the sentences A and B for each pre-training example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext).


```

1 for e in range(10):
2     print(f"e = {e}")
3     for _, data in enumerate(train_loader):
4         sentences : torch.Tensor = data["bert_input"]
5         for sentence in sentences:
6             print(sentence)
7             print(reconstruct(sentence))
8             print("\n")

```

e = 0

```

tensor([[ 3, 248,  4,  1, 63, 8152, 2471,  4, 63,  1, 8066, 189,
        6423,  4,  1,  1, 193,  1,  1,  2,  4, 179, 7948,  1,
        170, 3981,  1, 202, 7127, 357,  1, 189,  1, 689,  4,  1,
        888, 331, 201, 1623, 1981,  1, 808,  1, 193, 9943,  2,  0,
         0,  0])

```

<eos> He <mask> <unk> a Houston hotel <mask> a <unk> Owen and developed <mask> <unk> <unk> in <unk> <unk> <eos> <mask> to financial <unk> the church <unk> be completed by <unk>

```

tensor([[ 3,  1, 5716,  1, 565, 3790, 3467,  1,  1, 5771,  1,  1,
         1,  1, 1509, 232, 63,  1,  4,  1, 189,  4,  4,  1,
        6793, 2303, 1442, 193,  4, 2198, 4243,  2,  1,  4,  4, 1028,
        9504,  4, 239, 509, 202, 1927, 5651, 241,  1, 179, 812, 1476,
        559, 170])

```

<eos> <unk> January <unk> after former President <unk> signed <unk> <unk> <unk> known as a <unk> <mask> <unk> and <mask> <mask> <unk> protested among others in <mas

```

tensor([[ 3, 9765,  1, 610, 358,  1, 357, 63,  1,  1,  4,  1,
        4576, 189,  4, 193,  4, 6855,  1,  2, 248, 201,  1, 179,
        1429, 193, 223, 1687, 244,  1, 5905, 6216, 5810,  1, 63,  1,
         4, 6623,  1, 5525, 179, 7286,  1, 189, 201,  1, 179,  4,
        808, 193])

```

<eos> resent <unk> has been <unk> by a <unk> <unk> <mask> <unk> member and <mask> in <mask> permission <unk> <eos> He was <unk> to death in his office at <unk> University cy De

```

tensor([[ 3,  1, 540,  1,  1, 6996, 3889, 170, 2512, 232, 170,  1,
        6204, 189, 1119, 7380,  1,  1, 4976, 4, 9239,  2,  1,  4,
         1, 189,  4, 9755,  1,  4, 357,  1,  1,  1,  1,  1,
         1,  4,  1, 189, 4058,  1,  2,  0,  0,  0,  0,  0,
         0,  0])

```

<eos> <unk> In <unk> <unk> Bart joined the company as the <unk> director and later Union <unk> <unk> president <mask> 2013 <eos> <unk> <mask> <unk> and <mask> Taylor <unk> <mas

```

tensor([[ 3, 1225, 8892,  4,  1, 268, 179,  1, 170,  1, 241, 2895,
         1,  4, 191, 964,  1,  1, 235,  1, 189, 2476, 3942, 5568,
         1,  1,  1,  4,  4,  2,  1, 621, 1351, 2198,  4,  1,
         1,  1,  1, 949, 292, 2198,  1,  1,  4, 315, 5464,  2,
         0,  0])

```

<eos> One option <mask> <unk> is to <unk> the <unk> with private <unk> <mask> of next <unk> <unk> for <unk> and public fashion shows <unk> <unk> <unk> <mask> <mask> <eos> <unk>

```

tensor([[ 3,  1, 540,  1, 239,  1, 209, 170, 1246, 191, 9122,  1,
        ...

```

Figure 12 - Preprocessed Data

In order to feed the sentences into the LM, 3 types of embedding is performed on the sentences are the results are added before they are fed into the LM.

- Token embedding : WordPiece embedding with 10K token vocabulary was used. To perform WordPiece embedding “BertWordPieceTokenizer ” package from hugging face was used.
- Segment embedding : There are just two vector representations in the Segment Embeddings layer. All tokens belonging to sentence 1 are assigned to the first vector (index 0), whereas all tokens belonging to sentence 2 are assigned to the second vector (index 1).
- Positional embedding : The absolute position embedding is used to model how a token at one position attends to another token at a different position.

To learn the above mentioned embedding, we created a learnable embedding layer by inheriting nn.Embedding class from PyTorch.

3.3.2 Implementation

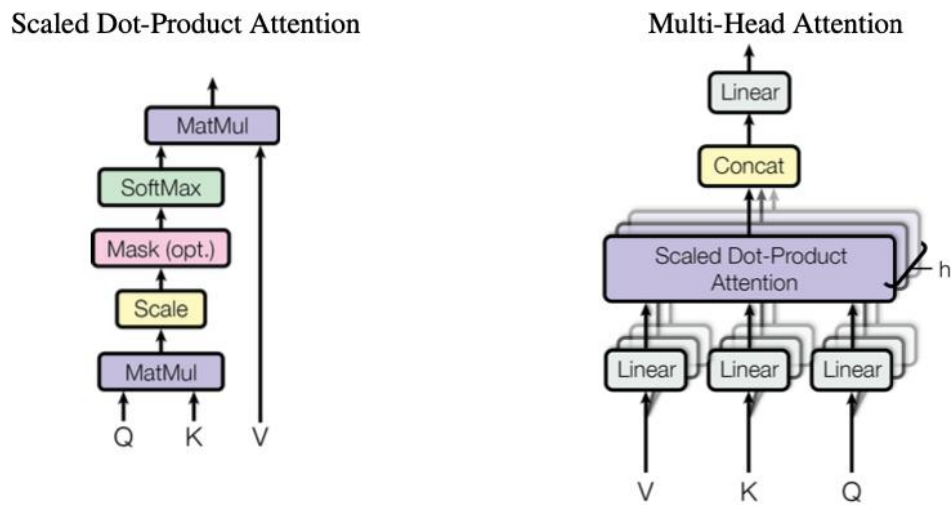


Figure 13 - Attention Architecture

BERT was built using a transformer's encoder. Rather than using existing packages, we implemented the transformer from scratch by implementing the paper “Attention Is All You Need”.

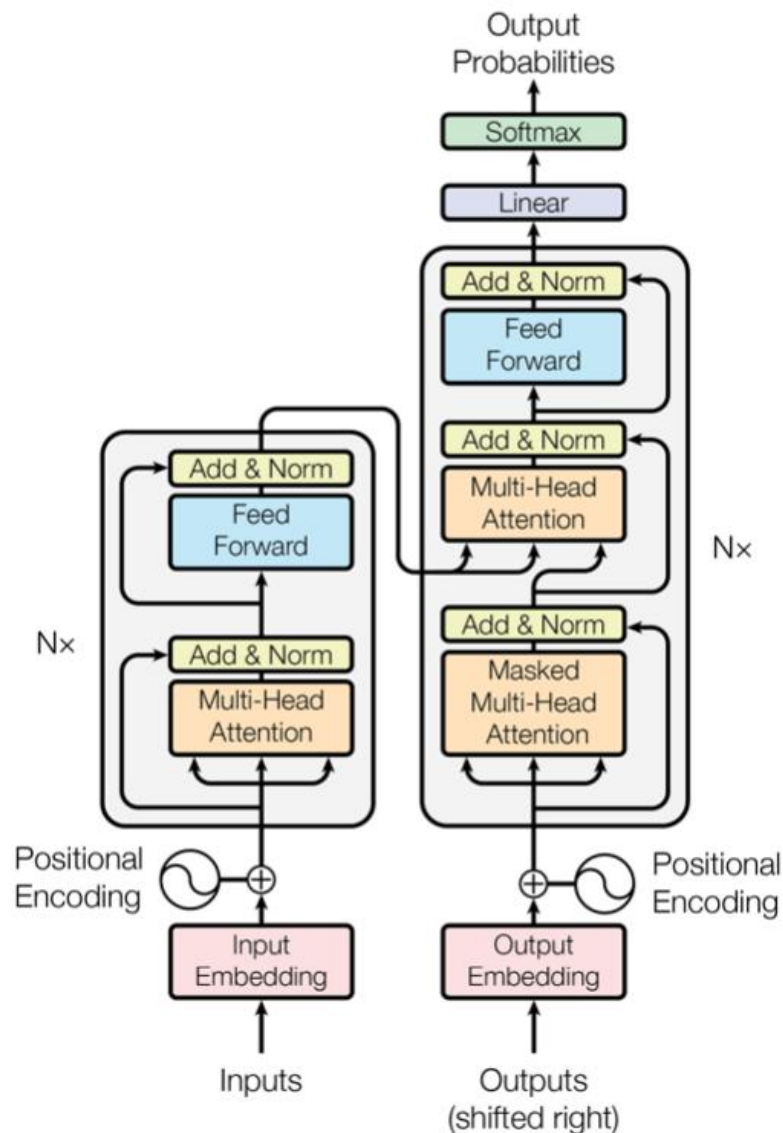


Figure 14 - Transformer Model

Below are the components of transformer explained :

- **Attention** : An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. [2]
- **Multi-headed attention** : Multi-head attention allows the model to jointly attend to information from different representation sub-spaces at different positions.

Each of the layers in our encoder and decoder of the transformer contains a fully connected feed-forward network, which is applied to each position separately and identically.

BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation in [2].

Transformer summary :

```
(encoder): Transformer(
  (layer): ModuleList(
    (0): BertLayer(
      (attention): Attention(
        (self): SelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): SelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (intermediate): Intermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
      (intermediate_act_fn): GELUActivation()
    )
    (output): Output(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
```

With our current limitation to GPU access, we implemented BERT small (L=12, H=768, A=12, Total parameters = 110M), where

L : Number of transformer blocks,

H : Hidden size,

A : Number of attention self-attention,

Our BERT was trained on a GPU provided by Google Colaboratory for 300 steps for an hour. It was able to achieve min(loss) of 3.05.

```

1 epochs : int = 1000
2
3 for epoch in range(epochs):
4     trainer.train(epoch)
5     trainer.test(epoch)

EP_train:298: 58% | 1/22 [00:00<00:08, 2.43it/s]{'epoch': 298, 'iter': 0, 'avg_loss': 4.921213150024414, 'avg_acc': 62.5, 'loss': 4.921213150024414}
EP_train:298: 50% | 11/22 [00:04<00:04, 2.66it/s]{'epoch': 298, 'iter': 10, 'avg_loss': 4.6673453070900655, 'avg_acc': 54.26136363636363, 'loss': 4.217276096343994}
EP_train:298: 95% | 21/22 [00:07<00:00, 2.67it/s]{'epoch': 298, 'iter': 20, 'avg_loss': 4.695810408819289, 'avg_acc': 52.38095238095239, 'loss': 5.138515472412109}
EP_train:298: 100% | 22/22 [00:08<00:00, 2.68it/s]
EP298_train, avg_loss= 4.689678365534002 total_acc= 52.028985507246375
EP_test:298: 100% | 1/1 [00:00<00:00, 15.51it/s]{'epoch': 298, 'iter': 0, 'avg_loss': 4.8163299560546875, 'avg_acc': 57.14285714285714, 'loss': 4.8163299560546875}
EP298_test, avg_loss= 4.8163299560546875 total_acc= 57.142857142857146

EP_train:299: 58% | 1/22 [00:00<00:08, 2.36it/s]{'epoch': 299, 'iter': 0, 'avg_loss': 4.717956066131592, 'avg_acc': 40.625, 'loss': 4.717956066131592}
EP_train:299: 50% | 11/22 [00:04<00:04, 2.65it/s]{'epoch': 299, 'iter': 10, 'avg_loss': 4.622108372774991, 'avg_acc': 46.30681818181818, 'loss': 4.753853797912598}
EP_train:299: 95% | 21/22 [00:07<00:00, 2.67it/s]{'epoch': 299, 'iter': 20, 'avg_loss': 4.7014105433509465, 'avg_acc': 47.61904761904761, 'loss': 5.177381992340088}
EP_train:299: 100% | 22/22 [00:08<00:00, 2.68it/s]
EP299_train, avg_loss= 4.708350636742332 total_acc= 47.68115942028985
EP_test:299: 100% | 1/1 [00:00<00:00, 17.01it/s]{'epoch': 299, 'iter': 0, 'avg_loss': 4.769043922424316, 'avg_acc': 42.857142857142854, 'loss': 4.769043922424316}
EP299_test, avg_loss= 4.769043922424316 total_acc= 42.857142857142854

EP_train:300: 58% | 1/22 [00:00<00:08, 2.38it/s]{'epoch': 300, 'iter': 0, 'avg_loss': 4.831128120422363, 'avg_acc': 28.125, 'loss': 4.831128120422363}
EP_train:300: 50% | 11/22 [00:04<00:04, 2.67it/s]{'epoch': 300, 'iter': 10, 'avg_loss': 4.775351264260032, 'avg_acc': 48.01136363636363, 'loss': 4.482288837432861}
EP_train:300: 95% | 21/22 [00:07<00:00, 2.65it/s]{'epoch': 300, 'iter': 20, 'avg_loss': 4.69798941839309, 'avg_acc': 47.470238095238095, 'loss': 4.945706367492676}
EP_train:300: 100% | 22/22 [00:08<00:00, 2.66it/s]
EP300_train, avg_loss= 4.70681001923301 total_acc= 47.10144927536232
EP_test:300: 100% | 1/1 [00:00<00:00, 16.63it/s]{'epoch': 300, 'iter': 0, 'avg_loss': 4.415318012237549, 'avg_acc': 42.857142857142854, 'loss': 4.415318012237549}
EP300_test, avg_loss= 4.415318012237549 total_acc= 42.857142857142854

EP_train:301: 58% | 1/22 [00:00<00:09, 2.31it/s]{'epoch': 301, 'iter': 0, 'avg_loss': 4.822868824005127, 'avg_acc': 40.625, 'loss': 4.822868824005127}
EP_train:301: 50% | 11/22 [00:04<00:04, 2.62it/s]{'epoch': 301, 'iter': 10, 'avg_loss': 4.721256646243009, 'avg_acc': 51.42045454545454, 'loss': 4.335842609405518}
EP_train:301: 95% | 21/22 [00:07<00:00, 2.66it/s]{'epoch': 301, 'iter': 20, 'avg_loss': 4.692597934177944, 'avg_acc': 50.44642857142857, 'loss': 5.102458477020264}
EP_train:301: 100% | 22/22 [00:08<00:00, 2.66it/s]
EP301_train, avg_loss= 4.68285624776657 total_acc= 50.869565217391305
EP_test:301: 100% | 1/1 [00:00<00:00, 17.02it/s]{'epoch': 301, 'iter': 0, 'avg_loss': 4.08685512542725, 'avg_acc': 78.57142857142857, 'loss': 4.08685512542725}
EP301_test, avg_loss= 4.08685512542725 total_acc= 78.57142857142857

EP_train:302: 58% | 1/22 [00:00<00:08, 2.38it/s]{'epoch': 302, 'iter': 0, 'avg_loss': 5.0967864990234375, 'avg_acc': 46.875, 'loss': 5.0967864990234375}

```

Figure 15 - BERT Training

3.3.3 Classifier

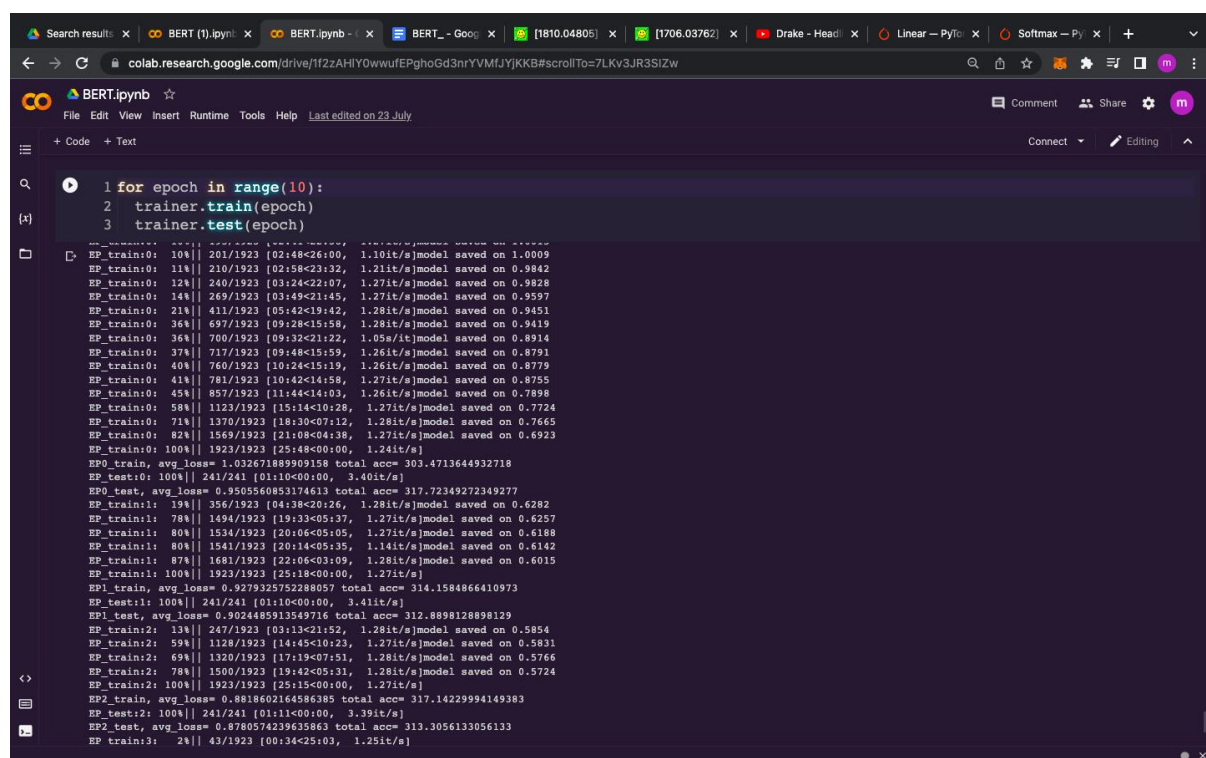
When we started to fine-tune the BERT from scratch, we ran into a problem. Tensor shape mismatch while feeding the output and label into the criterion. Due to time constraints, we left this problem unsolved. Therefore we used pre-trained BERT from hugging face and fine-tuned it to classify sentences into hate speech, offensive or normal.

We used a hate speech data-set from hugging face to train this classifier. Pre-processing mention in Data processing were used (WordPiece tokenization, position embedding and segment embedding)

The linear stack followed by BERT layer contains the following :

- Dropout layer : Linear layer with 768 input features and 3 output features to perform linear transformation.
- Softmax to rescale the tensor into n-dimensional output Tensor lies in the range $[0,1]$ and sums to 1.

The BERT classifier was fine-tuning for 3 epochs on the same GPU described in the above section.



```
1 for epoch in range(10):
2     trainer.train(epoch)
3     trainer.test(epoch)

EP_train:0: 100%| 241/241 [01:10:00:00, 3.40it/s] total acc= 317.72349272349277
EP_train:0: 11%| 210/1923 [02:58:23:32, 1.21it/s]model saved on 0.9842
EP_train:0: 12%| 240/1923 [03:24:22:07, 1.27it/s]model saved on 0.9828
EP_train:0: 14%| 269/1923 [03:49:21:45, 1.27it/s]model saved on 0.9597
EP_train:0: 21%| 411/1923 [05:42:19:42, 1.28it/s]model saved on 0.9451
EP_train:0: 36%| 697/1923 [09:28:15:58, 1.28it/s]model saved on 0.9419
EP_train:0: 36%| 700/1923 [09:32:21:22, 1.05it/s]model saved on 0.8914
EP_train:0: 37%| 717/1923 [09:48:15:59, 1.26it/s]model saved on 0.8991
EP_train:0: 40%| 760/1923 [10:24:15:19, 1.26it/s]model saved on 0.8779
EP_train:0: 41%| 781/1923 [10:42:14:58, 1.27it/s]model saved on 0.8755
EP_train:0: 45%| 857/1923 [11:44:14:03, 1.26it/s]model saved on 0.7898
EP_train:0: 58%| 1123/1923 [15:14:10:28, 1.27it/s]model saved on 0.7724
EP_train:0: 71%| 1370/1923 [18:30:07:12, 1.28it/s]model saved on 0.7665
EP_train:0: 82%| 1569/1923 [21:08:04:38, 1.27it/s]model saved on 0.6923
EP_train:0: 100%| 1923/1923 [22:48:00:00, 1.24it/s]
EP0_train, avg_loss= 1.032671889909158 total acc= 303.4713644932718
EP0_test: 100%| 241/241 [01:10:00:00, 3.40it/s]
EP0_test, avg_loss= 0.9505560853174613 total acc= 317.72349272349277
EP_train:1: 19%| 356/1923 [04:38:20:26, 1.28it/s]model saved on 0.6282
EP_train:1: 78%| 1494/1923 [19:33:05:37, 1.27it/s]model saved on 0.6257
EP_train:1: 80%| 1534/1923 [20:06:05:05, 1.27it/s]model saved on 0.6188
EP_train:1: 80%| 1541/1923 [20:14:05:35, 1.14it/s]model saved on 0.6142
EP_train:1: 87%| 1681/1923 [22:06:03:09, 1.28it/s]model saved on 0.6015
EP_train:1: 100%| 1923/1923 [25:18:00:00, 1.27it/s]
EP1_train, avg_loss= 0.9279325752288057 total acc= 314.1584866410973
EP1_test: 100%| 241/241 [01:10:00:00, 3.41it/s]
EP1_test, avg_loss= 0.9024485913549716 total acc= 312.8898128898129
EP_train:2: 13%| 247/1923 [03:13:21:52, 1.28it/s]model saved on 0.5854
EP_train:2: 59%| 1128/1923 [14:45:10:23, 1.27it/s]model saved on 0.5831
EP_train:2: 69%| 1320/1923 [17:19:07:51, 1.26it/s]model saved on 0.5766
EP_train:2: 78%| 1500/1923 [19:42:05:31, 1.28it/s]model saved on 0.5724
EP_train:2: 100%| 1923/1923 [25:15:00:00, 1.27it/s]
EP2_train, avg_loss= 0.8818602164586385 total acc= 317.14229994149383
EP2_test: 100%| 241/241 [01:11:00:00, 3.39it/s]
EP2_test, avg_loss= 0.8780574239635863 total acc= 313.3056133056133
EP_train:3: 2%| 43/1923 [00:34:25:03, 1.25it/s]
```

Figure 16 - BERT Classifier Training

3.4 API Integration

The model is deployed on an EC2 instance successfully and the API can be built in the instance itself using python flask library. Since the probabilities of hate speech and normal content is the only data returned by the model, a simple GET request from the created API with the content of a text message as input, will be sufficient.

Any application or platform can make use of this API to implement moderation. As an example, we've made use of the Discord.js library to create a bot that can perform auto moderation on messages being sent in a server. The Discord API sends info about event the bot can possibly can listen to : message creation, message edits, nickname changes and so on.

When the bot receives a message create event, it could send the content of the message to the API and receive the probabilities as JSON. The bot can make use of the response to either log the offensive content (so moderators can review) or delete the message by itself, if given the permissions to do. This way cyberbullying can be checked in a discord server through a discord bot.

4. Technology used

4.1 Git and GitHub

Git is a distributed version-control system. Version Control Systems are the software tools for tracking/managing all the changes made to the source code during the project development. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere. This tutorial teaches you GitHub essentials like repositories, branches, commits, and Pull Requests.

4.2 PyTorch

PyTorch is an open source machine learning library used for developing and training neural network based deep learning models. It is primarily developed by Facebook's AI research group. PyTorch can be used with Python as well as a C++. Naturally, the Python interface is more polished. Pytorch (backed by biggies like Facebook, Microsoft, Salesforce, Uber) is immensely popular in research labs. Not yet on many production servers - that are ruled by frameworks like TensorFlow (Backed by Google) -Pytorch is picking up fast.

4.3 Amazon EC2

Amazon Elastic Compute Cloud (EC2) is a part of Amazon.com's cloud-computing platform, Amazon Web Services (AWS), that allows users to rent virtual computers on which to run their own computer applications. EC2 encourages scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image (AMI) to configure a virtual machine, which Amazon calls an "instance", containing any software desired. A user can create, launch, and terminate server-instances as needed, paying by the second for active servers - hence the term "elastic". EC2 provides users with control over the geographical location of instances that allows for latency optimization and high levels of redundancy.

4.4 AWS Lake Formation

AWS Lake Formation is a service that makes it easy to set up a secure data lake in days. A data lake is a centralized, curated, and secured repository that stores all your data, both in its original form and prepared for analysis. A data lake lets you break down data silos and combine different types of analytics to gain insights and guide better business decisions.

5. Results and Output

5.1 Data analysis on Cyberbullying

We conducted our own analysis of cyberbullying using a data-set from kaggle. Gaussian Naive Bayes, Multinomial Naive Bayes and K neighbors classifier were training on the data-set. The following results shown in Fig. 17 and Fig. 18 were obtained :

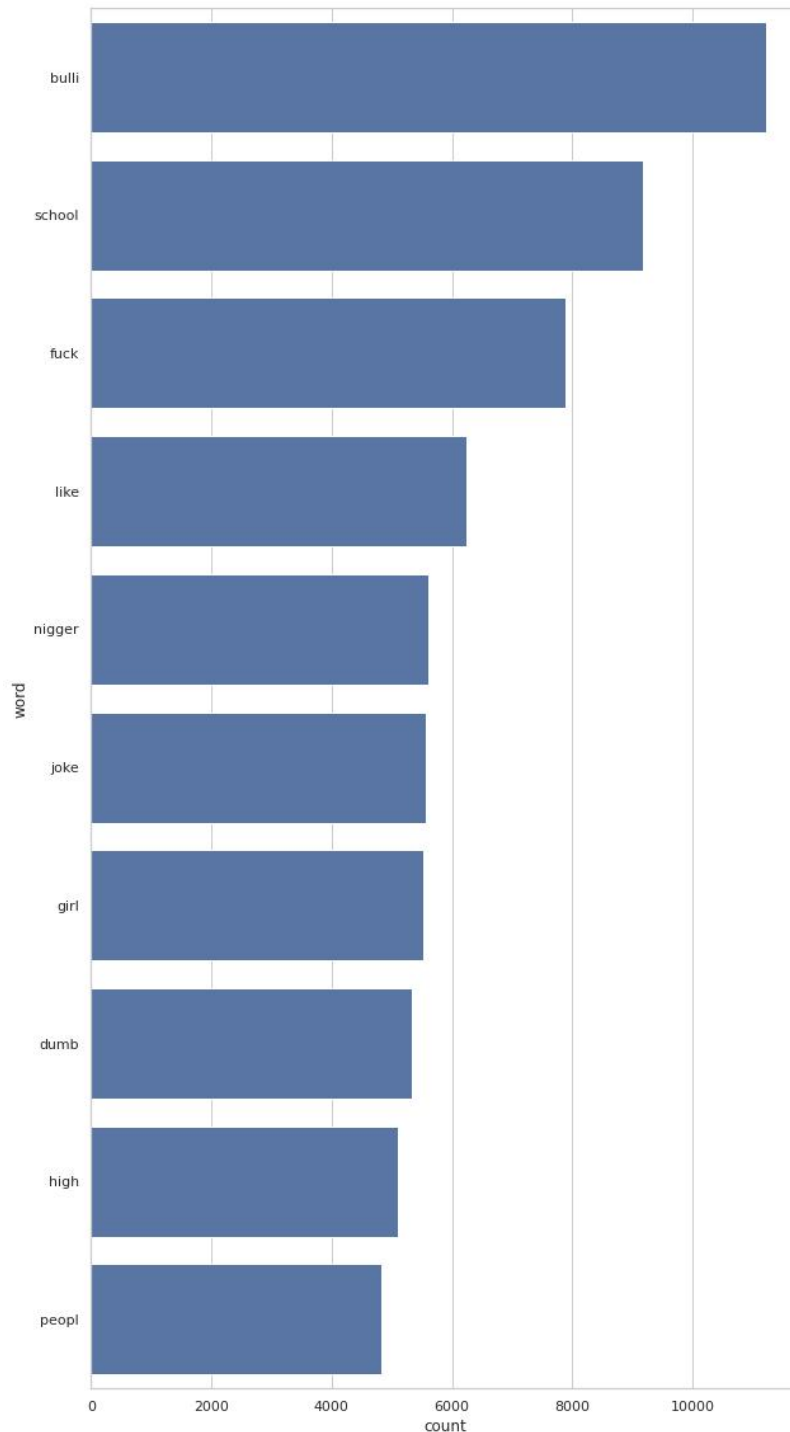


Figure 17 - Most Used Hate Words

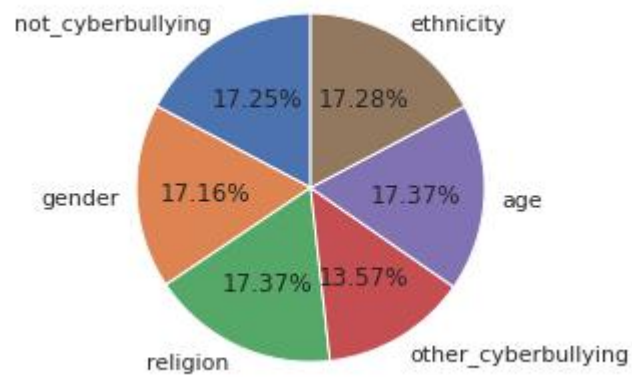


Figure 18 - Forms of Cyberbullying

5.2 Model training results

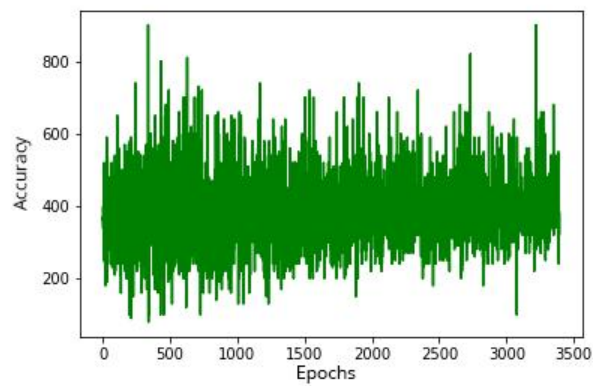


Figure 19 - Classifier Accuracy

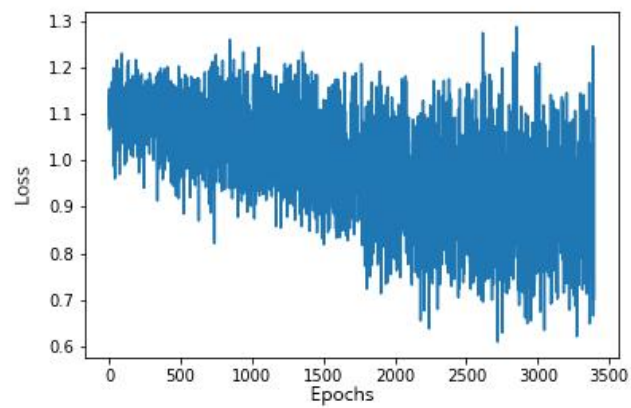


Figure 20 - Classifier Loss

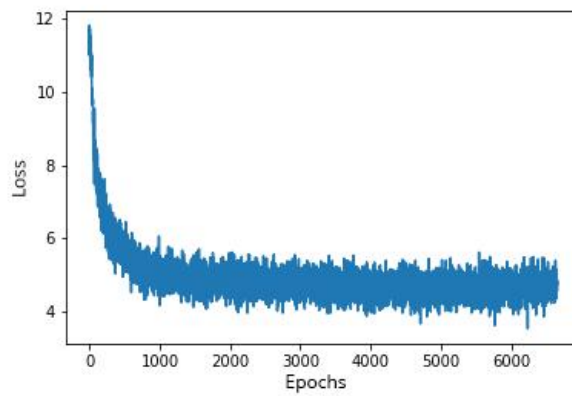


Figure 21 - Language Model Loss

5.3 API Integration Example with Discord.js

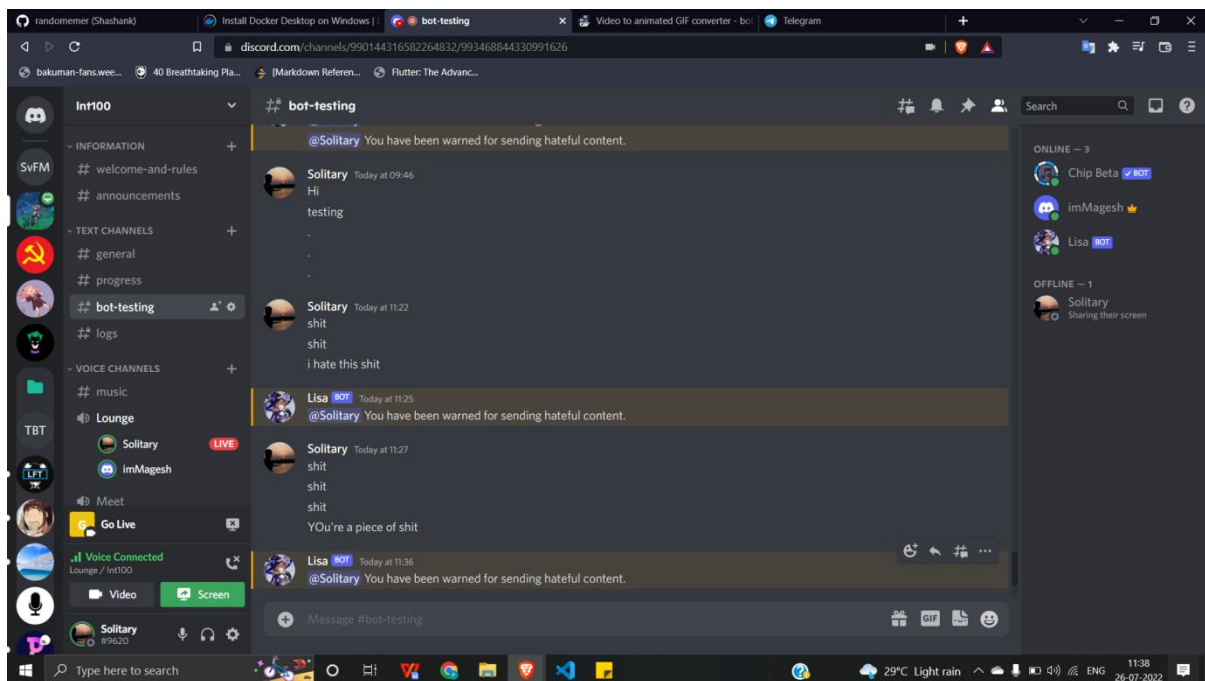


Figure 22 - Deleting messages and warning users

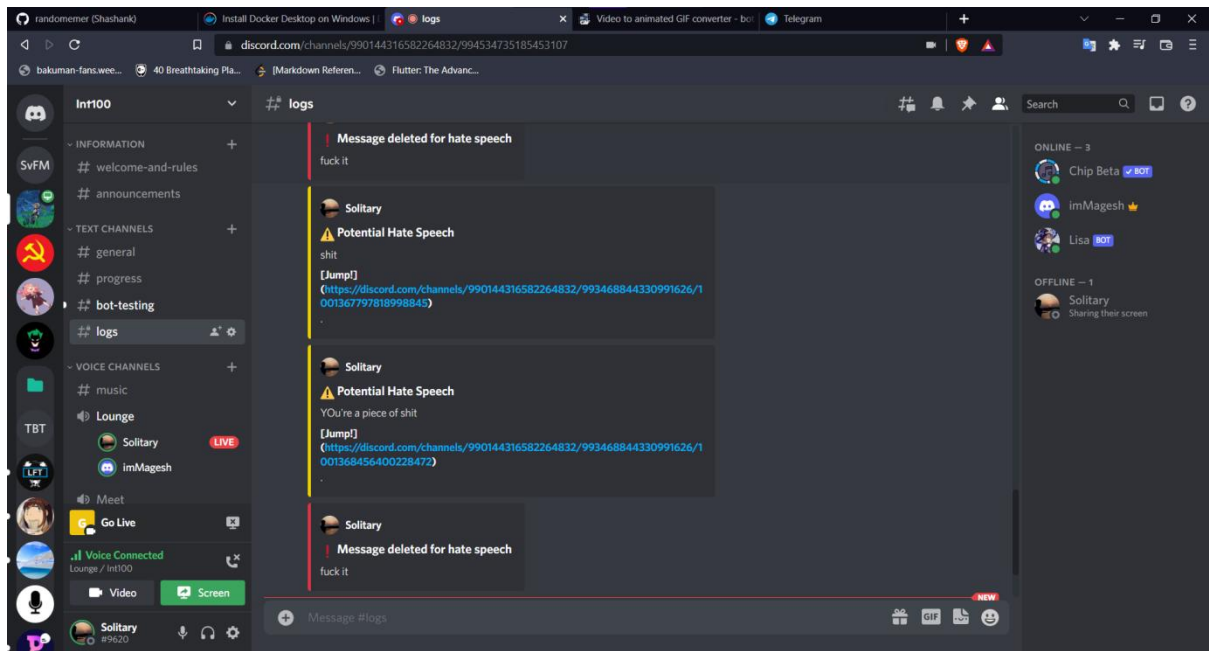


Figure 23 - Logging Cyberbullying related events

6. Timeline

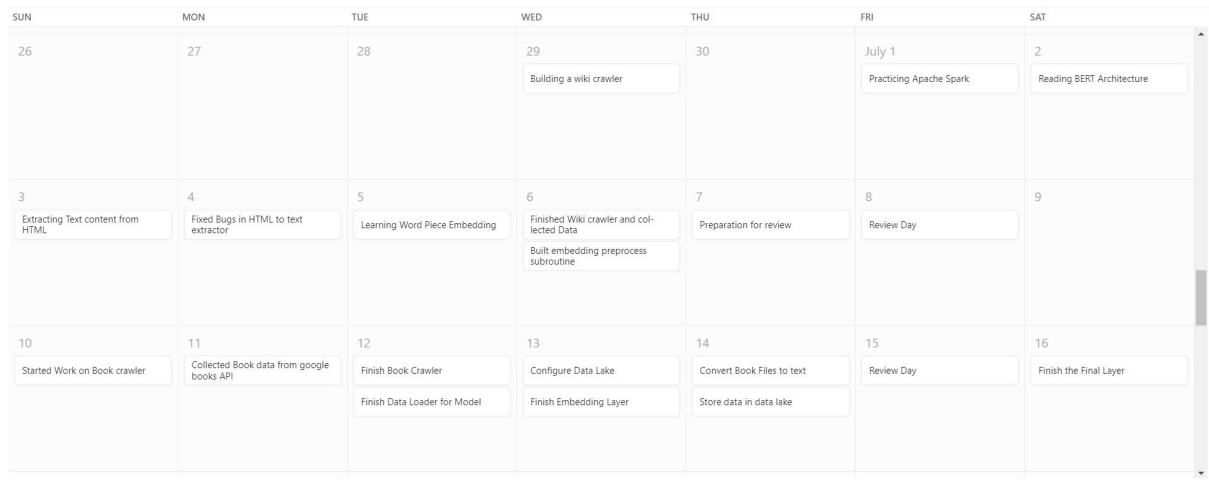


Figure 24 - Timeline June 29 - July 16

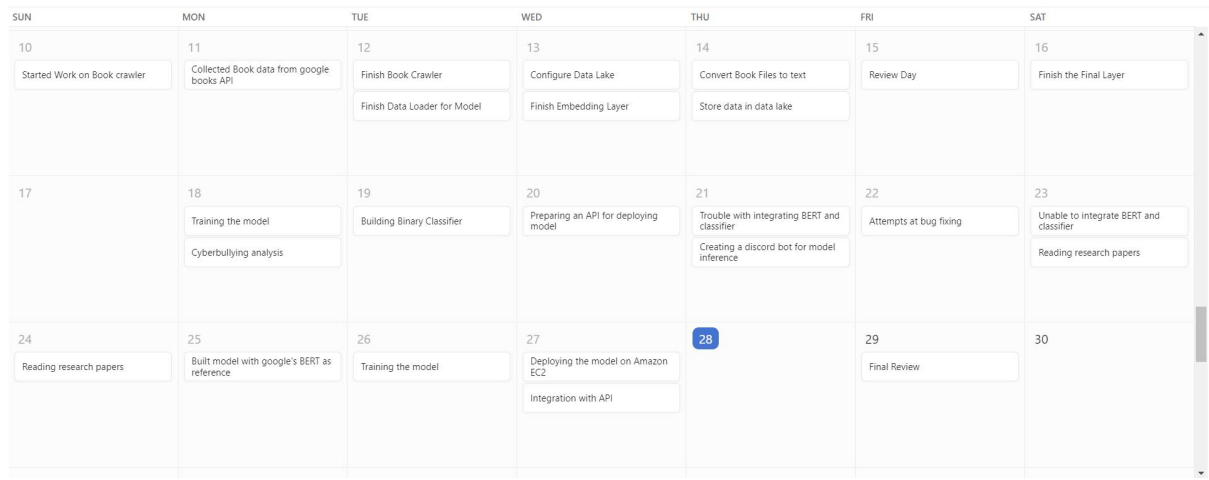


Figure 25 - Timeline July 10 - July 28

7. Scope for further enhancement

There's always scope for improvement, thus there are ways to improve the performance and reliability of the model in real situations.

Firstly, training the model with more data will result in an accurate model when compared to those with weak training and fewer epochs. More data could be collected for this purpose. The crawlers could be allowed to crawl for longer periods of time, messages from platforms like discord and twitter could be collected and utilize data-sets which are available.

Secondly, the model integration with the classifier could be improved to obtain accurate and precise outputs. The model's training could also be improved by increasing the number of parameters. As mentioned earlier, google's BERT has a very large number of parameters. However, we should keep in mind that requires intense compute and GPU, hence they might have to procured by renting GPU online / buying physically.

Thirdly, the model could be also be extended to handle images and perhaps videos through computer vision for detection of cyberbullying. This will be truly resourceful for social media platforms as 3.2 billion images and 720,000 hours of video are shared online daily.

Conclusion

Cyberbullying is a prevalent threat on the internet and can be detrimental to the psychological health of those are victims of it. With the advent of technology and development, Cyberbullying can be combated through machine learning and artificial intelligence.

We built a model with BERT and classifier to detect hate speech in text content. This model was deployed to Amazon EC2 along with an API to access the model. This model can be integrated in application through the API to keep cyberbullying in check.

Through hate speech detection, it's easier to tackle cyberbullying taking place in online social interactions. It helps keep discussions and conversations civil and prevents the harmful effects of online cyberbullying which can possibly cause damage to mental health. This will ultimately make online social platforms a little bit safer for everyone.

Providing a common way to make use of the machine learning model will make it easily accessible to anyone who wishes to integrate this in their projects. It's taking a step towards a safer and better internet.

References

1. <https://www.unicef.org/end-violence/how-to-stop-cyberbullying>
2. <https://blog.ipleaders.in/cyber-bullying-complete-analysis/>
3. <https://www.kaggle.com/datasets/andrewmvd/cyberbullying-classification>
4. <https://www.wikipedia.org>
5. <https://discord.com/developers/docs/reference>
6. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding : <https://arxiv.org/abs/1810.04805>
7. Attention Is All You Need : <https://arxiv.org/abs/1706.03762>
8. PyTorch : <https://pytorch.org/docs/stable/index.html>