

MULTILINGUAL PROFANITY DETECTION API USING DEEP LEARNING

By

Lim Dao Ern

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2022

REPORT STATUS DECLARATION FORM

Title: _____
Multilingual Profanity Detection API using Deep Learning

Academic Session: _____
June 2022

I _____
LIM DAO ERN
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.



(Author's signature)

Verified by,



(Supervisor's signature)

Address:

2278, Jalan Seksyen 2/8, _____

Taman Bandar Barat, _____

31900, Kampar, Perak _____

Jasmina Khaw Yen Min

Supervisor's name

Date: _____
9th September 2022

Date: _____
9 September 2022

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 9th September 2022

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that **Lim Dao Ern** (ID No: **19ACB06133**) has completed this final year project entitled "**Multilingual Profanity Detection API using Deep Learning**" under the supervision of **Dr. Jasmina Khaw Yen Min** (Supervisor) from the Department of **Computer Science**, Faculty of **Information and Communication Technology**.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.


Yours truly,



Lim Dao Ern

DECLARATION OF ORIGINALITY

I declare that this report entitled “**MULTILINGUAL PROFANITY DETECTION API USING DEEP LEARNING**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  _____

Name : Lim Dao Ern

Date : 9th September 2022

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr. Jasmina Khaw Yen Min who has given me this bright opportunity to engage in the profanity detection project. It is my first step to establish a career in deep learning field. A million thanks to you.

To a very special person in my life, Dr. Jasmina Khaw Yen Min, for her patience, unconditional support, and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

Profanity is an offensive use of language that is deemed impolite and rude. Profanity on the Internet is often associated with abusive intentions to cause psychological harm. A study shows that there has been a 70% increase in hate speech among teens since the COVID-19 lockdown began [6]. To counter the increasing prevalence of profanity in digital spaces, automated profanity detection systems can be used. However, most of the existing profanity detection systems used the list-based approach, which is flawed. The user can deliberately alternate the spellings of profane words to bypass the detection. Even with some of them using the machine learning approach, they are still far from perfect as they are not able to understand the context of words used. Therefore, this project proposes a deep learning model to detect profanity in text messages. The proposed model is capable of supporting multiple languages, understanding the context of words, and recognizing alternating spellings of profanity in text messages. Moreover, the model will be deployed as a RESTful API with 2 endpoints so that it can be implemented in other applications and websites easily. Apart from the multilingual profanity detection model, this project also contributes by providing the source code of the proposed system. The system will be written as an open-source project, anyone on the Internet who is interested in profanity detection can view the source code freely.

TABLE OF CONTENTS

TITLE PAGE	i
REPORT STATUS DECLARATION FORM	ii
FYP THESIS SUBMISSION FORM	iii
DECLARATION OF ORIGINALITY	iv
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	xi
LIST OF TABLES	xiv
LIST OF ABBREVIATIONS	xv
CHAPTER 1: INTRODUCTION	1
1.1 Background Information	1
1.2 Problem Statement and Motivation.....	2
1.2.1 Alternate Spellings of Words in Profanity Detection	2
1.2.2 Context of Words used in the Sentence	3
1.2.3 Lack of Multilingual Support for Profanity Detection	3
1.3 Motivation	3
1.4 Project Scopes	5
1.5 Project Objectives	6
1.6 Contributions.....	6
1.7 Report Organization	6
CHAPTER 2: LITERATURE REVIEWS	8
2.1 Previous Studies in Profanity Detection.....	8

2.1.1	Multilingual Offensive Language Identification with Cross-lingual Embeddings	8
2.1.2	Method of Profanity Detection Using Word Embedding and LSTM.....	11
2.1.3	Automated Hate Speech Detection and the Problem of Offensive Language ...	13
2.2	Existing Profanity Detection System	15
2.2.1	Profanity-check	15
2.3	Critical Remarks of Previous Studies and Existing System.....	17
CHAPTER 3: METHODOLOGY		18
3.1	Methodologies.....	18
3.1.1	Dataset Acquisition.....	18
3.1.2	Model Training and Testing.....	18
3.1.3	API Implementation.....	18
3.2	Tools.....	19
3.2.1	Hardware.....	19
3.2.2	Software	19
3.2.3	Services.....	20
3.3	Timeline	21
CHAPTER 4: SYSTEM MODEL		23
4.1	LSTM with FastText Embeddings	23
4.2	Bidirectional Encoder Representations from Transformers.....	26
CHAPTER 5: SYSTEM DESIGN		29
5.1	System Block Diagram.....	29
5.2	PyPI Package (deep-profane).....	30

5.3	RESTful API (deep-profane-rest)	32
5.4	Web Chatting Application (DeepProfane Chat).....	33
CHAPTER 6: SYSTEM IMPLEMENTATION		36
6.1	Software Setup	36
6.2	Service Setup.....	36
6.3	Implementation of Profanity Detection Model	36
6.3.1	Datasets Acquisition	36
6.3.2	Datasets Exploration	38
6.3.3	Training of LSTM Model	40
6.3.4	Training of BERT Model.....	43
6.4	Development of DeepProfane Package.....	45
6.4.1	Model Hosting	45
6.4.2	Packaging as Python Package (deep-profane)	46
6.4.3	Serving of RESTful API (deep-profane-rest)	48
6.4.4	Deployment of Web Chatting Application (DeepProfane Chat)	50
6.5	System Operation	54
6.6	Implementation Issues and Challenges	58
CHAPTER 7: SYSTEM EVALUATION		59
7.1	Performance Metrics	59
7.1.1	Accuracy	59
7.1.2	Precision, Recall, and F1-Score	59
7.1.3	Precision-Recall Curve	60
7.1.4	Receiver Operating Characteristic Curve	61
7.2	Testing Result of the Proposed Model	61
7.3	Objectives Evaluation	64

7.4	Concluding Remarks	65
CHAPTER 8: CONCLUSION		66
8.1	Conclusion.....	66
8.2	Contributions.....	67
8.3	Future Works.....	67
REFERENCES		68
APPENDIX.....		A-1
POSTER.....		A-4
PLAGARISM CHECK RESULT.....		A-5
FYP 1 CHECKLIST		A-7

LIST OF FIGURES

Figure Number	Title	Page
Figure 1-1	Profanity filter in action on Discord.	1
Figure 1-2	Overview of the project scope split into research and development based.	5
Figure 2-1	XLM-R transformer text classification architecture.	8
Figure 2-2	Transfer learning strategy.	9
Figure 2-3	Overview of the system architecture.	12
Figure 2-4	CNN, GRU, and LSTM model results.	13
Figure 2-5	True versus predicted categories.	15
Figure 2-6	Usage of profanity-check with Jupyter Notebook.	16
Figure 2-7	Failed profanity and non-profanity classification by profanity-check.	17
Figure 3-1	Gantt chart for the timeline of project 1.	21
Figure 3-2	Gantt chart for the timeline of project 2.	22
Figure 4-1	Architecture of the LSTM profanity detection model.	23
Figure 4-2	Example word embeddings.	24
Figure 4-3	A single LSTM Cell	25
Figure 4-4	Input tokens and output tokens of BERT.	26
Figure 4-5	Architecture of profanity detection model based on BERT.	27
Figure 5-1	Block diagram of the DeepProfane package, which has 3 modules, including PyPI package, RESTful API, and a web app.	29
Figure 5-2	Overview class diagram for deep-profane package.	31
Figure 5-3	Architecture for deep-profane-rest, profanity detection RESTful API.	32
Figure 5-4	Simplified block diagram for DeepProfane Chat module.	33
Figure 5-5	Entity relation diagram for the database of DeepProfane Chat.	35
Figure 6-1	Source code to download and unzip Jigsaw Multilingual Toxic Comment Classification dataset from Kaggle.	37

Figure 6-2	Preview of the train set of Multilingual Toxic Comment Classification.	38
Figure 6-3	Distribution and box plot of words count for all text in the train set.	38
Figure 6-4	Preview of the test set of Multilingual Toxic Comment Classification.	39
Figure 6-5	Source code for tokenizing the input text corpus in the notebook document.	40
Figure 6-6	Source code for padding the tokenized text corpus.	40
Figure 6-7	Source code for downloading FastText embeddings and transforming the words (tokens) into embedding vectors.	41
Figure 6-8	Source code for constructing the proposed LSTM profanity detection model.	42
Figure 6-9	Source code for constructing BERT profanity detection model.	43
Figure 6-10	Converting dataset in Pandas DataFrame to TensorFlow Dataset.	43
Figure 6-11	Model summary by TensorFlow for the proposed BERT profanity detection model.	44
Figure 6-12	Trained model hosted using GitHub Releases.	45
Figure 6-13	The file structure of the deep-profane, Python package.	46
Figure 6-14	Source code for ProfanityValidator in Python.	47
Figure 6-15	The Python package, deep-profane, published at PyPI.	48
Figure 6-16	Source code, app.py for deep-profane-rest in Python, by using flask, flask_restful.	48
Figure 6-17	Screenshot of the running of the deep-profane-rest on Vultr server.	50
Figure 6-18	The simplified file structure of the deep-profane-chat.	51
Figure 6-19	The schema definition inside 'schema.prisma' file.	51
Figure 6-20	Heroku Postgres for deep-profane-chat.	52
Figure 6-21	Specifying Google OAuth Provider by using NextAuth.js.	52
Figure 6-22	Setting up of OAuth 2.0 client for DeepProfane Chat at Google Cloud Console.	53

Figure 6-23	Code snippet to perform profanity detection using deep-profane-rest.	53
Figure 6-24	Overview of DeepProfane Chat web app on Heroku dashboard.	54
Figure 6-25	Usage of deep-profane module.	55
Figure 6-26	The request and response of deep-profane-rest for profane_prob endpoint.	55
Figure 6-27	The request and response of deep-profane-rest for is_profane endpoint.	56
Figure 6-28	The comments board of DeepProfane Chat embedded inside a website.	56
Figure 6-29	Profanity detection in action in DeepProfane Chat.	57
Figure 6-30	Admin dashboard for the comment board shown in Figure 6-28.	57
Figure 7-1	Example precision-recall curve.	60
Figure 7-2	Example of Receiver Operating Characteristic curve.	61
Figure 7-3	Confusion matrix for LSTM model.	63
Figure 7-4	Precision-recall curve for LSTM model.	63
Figure 7-5	ROC curve for LSTM model.	63
Figure 7-6	Output of the English BERT model for the sentences ‘you are suck’ and ‘i suck my thumb’.	64
Figure 7-7	Output of the Multilingual BERT model with the first input as English profane text, second as Russian profane text, and third as English non-profane text.	64

LIST OF TABLES

Table Number	Title	Page
Table 1-1	Two sentences using the word ‘suck’.	3
Table 1-2	Reporting structure.	7
Table 2-1	Results ordered by macro (M) F1 for Bengali and weighted (W) F1 for Hindi and Spanish.	11
Table 2-2	BoW representation for ‘the ball drops into the hole’	16
Table 3-1	Table 3-1: Specification of laptop.	19
Table 4-1	Example tokenization process.	24
Table 5-1	Sample request and responses of Profanity Detection REST API.	32
Table 6-1	Language code and its name for all the supported languages in the test set.	39
Table 6-2	Training hyperparameters for the LSTM model.	42
Table 6-3	Training parameters of BERT profanity detection model.	45
Table 7-1	Performance of LSTM, English BERT, and Multilingual BERT on the test set.	62

LIST OF ABBREVIATIONS

<i>ITU</i>	International Telecommunication Union
<i>LSTM</i>	Long short-term memory
<i>PyPI</i>	Python Package Index
<i>UNICEF</i>	United Nations Children’s Fund
<i>XLM</i>	Cross-lingual language model
<i>BERT</i>	Bidirectional Encoder Representations from Transformers
<i>CNN</i>	Convolutional neural network
<i>GRU</i>	Gated recurrent unit
<i>SVM</i>	Support Vector Machines
<i>JSON</i>	JavaScript Object Notation
<i>IDE</i>	Integrated development environment

CHAPTER 1: INTRODUCTION

1.1 Background Information

Profanity is an offensive use of language that is deemed impolite and rude. Sometimes, it is also known as cursing, or swearing. Profanity can be viewed as an extreme way to express disrespectful feelings towards something by using abusive language. In most cases, bad words or expletives are used to make a message profane. For each language, there are usually sets of words which are widely recognized by the speaker as expletive. For example, in English, some of the common expletives include “f*ck”, “b*tch”, “c*nt”, etc. By making use of these words, one can construct messages to insult and bully in a verbal manner.

Although it is possible for profanity to be used as a signal of informality or close relationship between the speaker and listener, social interactions on digital public platforms usually involve netizens who have distant or no relationship with each other [1]. Therefore, profanity on the Internet is often associated with abusive intention to offend and cause emotional or psychological harm.

To counter the increasing prevalence of profanity in digital spaces, content moderation is one of the solutions to this problem. Content moderation is the practice of screening user-generated content which could be presented in a wide range of forms like images, videos, posts, messages, and more. In this paper, monitoring of text messages for profanity is specifically discussed. Profanity detection is a subset of content moderation which allows the system to classify if a message is offensive or not, so that appropriate action could be taken like hiding or censoring the message. Profanity detection not only can be applied to social networks, but also other platforms like community forums (Yin et al., 2009), in-game chatting systems, and sites with comment streams. Figure 1-1 shows a profanity filter blocking a profane message from being sent to public chat on a messaging software, Discord.

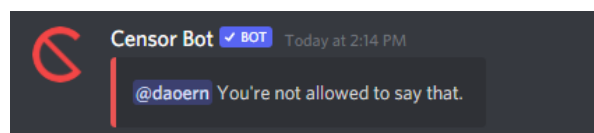


Figure 1-1: Profanity filter in action on Discord.

In general, content moderation can be categorized into manual moderation and automated moderation, or both. Most of the social networks adopt a combination of automated filtering and human moderators to keep unwanted content off their platforms [3].

For human moderation, it employs humans to manually monitor and screen the content generated by the users. Human moderation may filter the content with high reliability and accuracy, but it has its own limitations. First, humans could fail to moderate in real-time like messages going through the online chatting systems. Secondly, it is not feasible to utilize human moderation when there is a sheer amount of content generated like huge social media sites. This is due to the fact that human moderators often work in highly stressful conditions, in which they have to click through hundreds of flagged contents per day [2].

Therefore, a better approach would be adopting automated moderation. Automated moderation refers to the screening using Artificial Intelligence (AI) without humans. For profanity detection, this means the text message can be fed through a program, then the program will automatically determine if the message is profane. There are multiple approaches to implement the program. One of the ways is to define a list of flagged words, then the program will determine the text message as profane if the message contains any one of the flagged words. An alternative is to use machine learning or deep learning, train a model to recognize profanity, and then implement it into the program.

1.2 Problem Statement and Motivation

1.2.1 Alternate Spellings of Words in Profanity Detection

Most of the existing profanity detection libraries use the list-based approach, which requires a wordlist to identify profanity. For Python, these libraries can be found on the official software repository, Python Package Index (PyPI). From PyPI, almost all of the profanity detection library use the list-based approach like ‘profanity’ by Ben Friedland (32-word wordlist), ‘better-profanity’ by Son Nguyen Thanh (140-word wordlist), and ‘profanityfilter’ by Areeb Beigh (418-word wordlist).

List-based approach is straightforward to implement, but it has its own limitations. First, it can be easily circumvented just by alternating the spellings of words. For instance, one can deliberately change the word ‘bitch’ to ‘b!tch’ or ‘b1tch’, and then the profanity filter will fail to detect profanity. This is because the algorithm of the list-based approach simply looks

for if there is any exact match of words inside the wordlist. This is an issue, as even with alternate spellings, the word still carries the same offensive meaning to humans.

1.2.2 Context of Words used in the Sentence

Secondly, list-based profanity detection libraries cannot capture the context in which the word is used in the sentence. Considering both of the sentences in Table 1-1, assume the word ‘suck’ is included in the wordlist, then the profanity filter will evaluate both sentences A and B as profanity, but obviously sentence A does not mean to be profane whatsoever.

Sentence A	I suck my thumb.
Sentence B	You are suck.

Table 1-1: Two sentences using the word ‘suck’.

Furthermore, list-based profanity will also fail if one tries to swear by using only permissible words. For example, ‘Your mom is a fat cow’ will pass the list-based profanity detection although it has serious profanity. This is because the wordlist could not include the word ‘fat’, nor ‘cow’ one of the flagged words.

1.2.3 Lack of Multilingual Support for Profanity Detection

Another limitation of existing profanity detection libraries is that they do not support multilingualism. Even with non list-based libraries like ‘profanity-filter’ by Roman Infiaskas, and ‘profanity-check’ by Victor Zhou, which use machine learning, they are still restricted to only one language - English. While there are a few studies published in Chinese [8], Arabic [9], and Greek [7], they are still monolingual as per paper at the end. They are still not a single system which can truly work with multiple languages.

1.3 Motivation

At the end of 2019, there were 4.0 billion Internet users worldwide, which is just more than half of the world population. Almost 70% of the world's youth (aged 15-24 years) are using the Internet, and virtually all young people in developed countries are using the Internet [5].

With the number of Internet users rising [5], the amount of offensive and abusive content related to cyberbullying will also grow. This is an issue especially with a huge

proportion of the world's youth using the Internet. A UNICEF poll found that more than 33% of young people in 30 countries said they have been a victim of online bullying, with 20% reporting having skipped school due to cyberbullying. Even worse, there has been a 70% increase in hate speech among teens since the COVID-19 lockdown began [6]. Cyberbullying exists in the form of text messages like hate speech, which usually includes profanity. This is due to the fact that profanity mostly indicates offensive, impolite and rude attitudes, making them an essential feature of cyberbullying. Therefore, profanity detection is important as it can be adopted as one of the strategies to combat cyberbullying.

Furthermore, as of March 2020, an estimate showed that only 25.9% Internet users speak English, followed by 19.4% for Chinese, and 7.9% for Spanish. So, profanity detection should support multiple languages in order to cover the diversity of the languages spoken on the Internet globally.

Apart from cyberbullying, the regulatory framework also drives the motivation of building a proper profanity detection system. The European Union (EU) law makes 4 types of content illegal, which includes xenophobic hate speech. In May 2016, the European Commission launched the Code of Conduct on countering illegal hate speech online. The companies are committed to remove content deemed to be illegal from their online platform in less than 24 hours. Thus, the site should take the responsibility to moderate content generated by their users proactively.

To conclude, content moderation is becoming more relevant with the increase of Internet users, and enforcement of laws and regulations. Thus, the motivation of profanity detection API is working towards creating a safer environment for all Internet users, regardless the language they speak.

1.4 Project Scopes

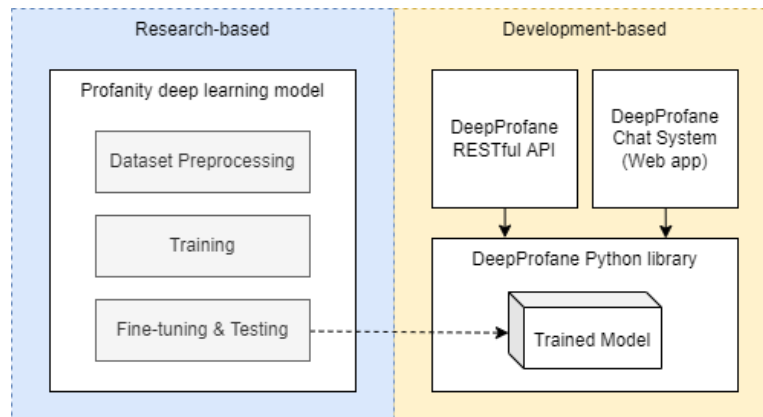


Figure 1-2: Overview of the project scope split into research and development-based.

This project is both research-based and development-based. The research part is to design and develop a profanity detection model that is capable of understanding alternative expletive spellings, context of the bad words, and also multiple languages. The datasets required for the training of the model will be collected from online sources, like Kaggle. Next, the datasets will be preprocessed before input to the model. The model will be evaluated and tested based on various performance metrics, like accuracy, recall, loss, etc. During Project 1, a simple functional model is trained and deployed as a proof of concept. Then in Project 2, multiple deep learning models will be thoroughly studied and proposed to select the best performing model for profanity detection. Lastly, the hyperparameters of the model will also be fine-tuned.

As for the development part of this project, the deliverable is a package called DeepProfane with a suite of tools including a Python package, a RESTful API, and a web chatting application. All these tools are intended to provide a way for end users to make use of the proposed profanity detection model. First, the model will be packaged as a PyPI Python library, the use case is an average developer can import this library and perform profanity detection tasks without any deep learning knowledge. Next, the RESTful API is also designed for developers. It contains 2 endpoints, one will return a boolean value stating if the message is profane, and another one will return the probability of profanity. The last module built for Internet users to fight against profane messages. It is a web chatting application in which the website owners can embed the chat box with profanity filtering enabled into their website, with the hope to fight against profane comments.

1.5 Project Objectives

The objectives of this project are summarized as the following list.

1. To develop a profanity detection algorithm using deep learning. Unlike list-based approach, the proposed algorithm has to be capable of recognizing alternate spellings of profanity, and capable of understanding the context of words used in the sentence.
2. To implement multilingualism in the proposed profanity detection model, supporting multiple languages other than English in a single model.
3. To implement the proposed profanity detection model as a RESTful API. Two endpoints are included, one for checking if the message is profane, another one for calculating the probability of profanity of the message.
4. To provide a chatting system with automatic profanity detection enabled that can be easily embedded into third-party websites.

1.6 Contributions

This project improves the method to perform profanity detection using deep learning instead of the list-based approach. The proposed method of profanity detection is expected to be more accurate than the list-based approach. Thus, this project provides digital platforms a more reliable way to moderate profanity within their user generated content. Moreover, the proposed profanity detection works with multiple languages, therefore covering the diversity of Internet users online who speak different languages. So, this project contributes by allowing those websites with less-common languages being able to moderate profanity automatically.

Furthermore, profanity detection API will be written as an open-source project. The source code will be publicly available to everyone interested in working in profanity detection. This provides developers with tools and resources to implement a better profanity detection system to their platform.

1.7 Report Organization

There are a total of 8 chapters in this report. The following table listed all the chapters and its description.

No	Chapter	Description
1	Introduction	<ul style="list-style-type: none"> • Provides background of the project. • Define problem statements, and objectives. • Discuss motivation and contribution.
2	Literature Review	<ul style="list-style-type: none"> • Perform preliminary studies on previous papers and existing systems. • Comparison between different technologies.
3	Methodology	<ul style="list-style-type: none"> • Define methods used to complete this project. • Specify the hardware, software, and services requirements. • Display the timeline of the project.
4	System Model	<ul style="list-style-type: none"> • Study and design profanity detection model by using deep learning.
5	System Design	<ul style="list-style-type: none"> • Design how the model deployed to end users. • Show the block diagram and discuss the key concept of the system. • Design tools like Python package, RESTful API, etc. • Focuses on the development part of this project.
6	System Implementation	<ul style="list-style-type: none"> • Document the implementation of the system, including source code snippet. • Discuss implementation challenges.
7	System Evaluation	<ul style="list-style-type: none"> • Explain performance metrics used. • Perform evaluation on the model. • Evaluate the project itself based on objectives.
8	Conclusion	<ul style="list-style-type: none"> • Discuss conclusion, novelties, contributions, and future work.

Table 1-2: Reporting structure.

CHAPTER 2: LITERATURE REVIEWS

The literature review is organized into 2 sections. First section discusses the previous studies in profanity detection, while the second section discusses existing profanity detection systems. At the end of this chapter, solutions to the problem statement are proposed based on the studies and systems reviewed.

2.1 Previous Studies in Profanity Detection

2.1.1 Multilingual Offensive Language Identification with Cross-lingual Embeddings

To increase the performance of the systems with non-English languages, data augmentation techniques and multilingual word embeddings have been used. However, state-of-the-art cross-lingual transformer models, XLM-R is not yet applied in offensive detection [7]. XLM-R has been trained on 104 languages and it proved to be highly performant in single language benchmarks while also recording an amazing result in multilingual benchmarks.

Ranasinghe and Zampieri therefore proposed to use multilingual word embeddings for offensive language detection. English data is used to perform predictions in 3 other languages including Bengali, Hindi, and Spanish. Furthermore, the proposed methods not only showed that they can predict for multiple languages, but also for various tasks.

Like other transformer model designs, XLM-R architecture is also suitable for text classification tasks. The diagram for the XLM-R transformer architecture classification task is shown in Figure 2-1.

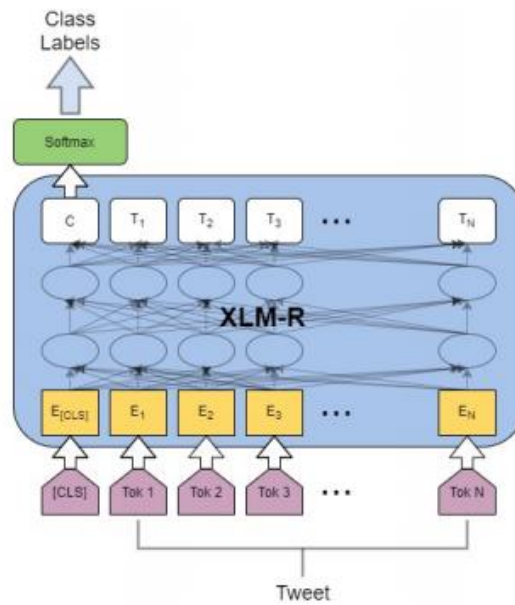


Figure 2-1: XLM-R transformer text classification architecture [7].

XLM-R-large model contains about 125 million parameters with 12 layers, 768 hidden-states, 3072 feed-forward hidden-states and 8 heads. It accepts a sequence of maximum 512 tokens as input. The first token is [CLS] which is used to represent special classification embedding. In the text classification task, the final hidden state of the first token will be taken as the representation for the whole message. Next, a softmax classifier as shown in Equation 2.1 is added on top of XLM-R which gives the probability of label, c , where W represents task-specific parameter matrix.

$$p(c|h) = \text{softmax}(Wh)$$

Equation 2-2: Softmax classifier for probability of label c [7].

Training a classification model with a language with large number of resources like English, using a cross-lingual transformer model, is the main concept of this paper. Then, the weights of the trained model are saved and later used to initialize the weights of the model during the training phase for the language with minimal number of resources. This is called transfer learning, and it is shown in Figure 2-2.

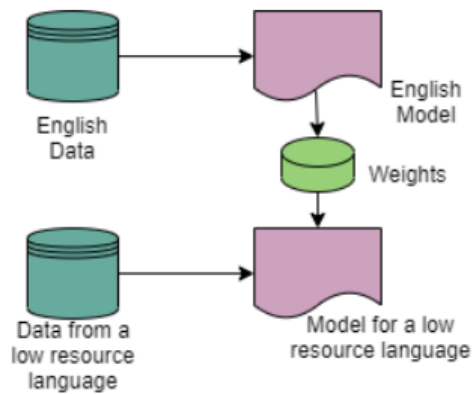


Figure 2-2: Transfer learning strategy [7].

Datasets are acquired for multiple languages, including English, Hindi, Spanish, and Bengali. These 4 datasets acquired haven been used in tasks in 2019 and 2021, thus performance of the proposed model can be compared to other approaches. For English dataset, Offensive Language Identification Dataset (OLID) is chosen because of its flexibility provided by hierarchical annotation model. The dataset is split with a training set to validation set ratio of 0.2. After fine tuning the hyperparameters, the learning rate of 1×10^{-5} and 3 epochs is configured for all languages. Training time for the English language used around 60 minutes and other languages used around 30 minutes.

The performance of the system is evaluated in terms of weighted and macro F1 score. The performance score of the proposed methodology which is XLM-R (TL) is compared with other models like BERT and XLM-R. The result is tabulated in Table 2-1, and it showed that XLM-R (TL) outperforms all of the other methods tested. TL indicates the model used transfer learning strategy as discussed previously.

Language	Model	M F1	W F1
Bengali	XLM-R (TL)	0.8415	0.8423
	Risch and Krestel (2020)	0.8219	
	BERT-m (TL)	0.8197	0.8231
	XLM-R	0.8142	0.8188
	BERT-m	0.8132	0.8157
	Baseline	0.2498	0.4991

Hindi	XLM-R (TL)	0.8568	0.8580
	BERT-m (TL)	0.8211	0.8220
	Bashar and Nayak (2019)	0.8149	0.8202
	XLM-R	0.8061	0.8072
	BERT-m	0.8025	0.8030
	Baseline	0.3510	0.3798
Spanish	XLM-R (TL)	0.7513	0.7591
	BERT-m (TL)	0.7319	0.7385
	Vega et al. (2019)		0.7300
	Perez and Luque ´ (2019)		0.7300
	XLM-R	0.7224	0.7265
	BERT-m	0.7215	0.7234
	Baseline	0.3700	0.4348

Table 2-1: Results ordered by macro (M) F1 for Bengali and weighted (W) F1 for Hindi and Spanish.[7].

2.1.2 Method of Profanity Detection Using Word Embedding and LSTM

Internet users are deliberately modifying the nucleus of a word to avoid detection by the profanity detection system. Currently, the profanity detection system has difficulties identifying permissible words which have similar sounding to a swear word. Therefore, Yi, Lim, Ko and Shin, proposed a method using FastText model and LSTM model to accurately identify profanity when the nucleus of the word is modified to prevent detection, and when word with similar morphology to expletive is used.

Figure 2-3 displays the system architecture of the proposed profanity detection method. It is generally separated into 2 parts, which are Training Data Process which trains the model, and Testing Data Process which detects profanity for a given text message.

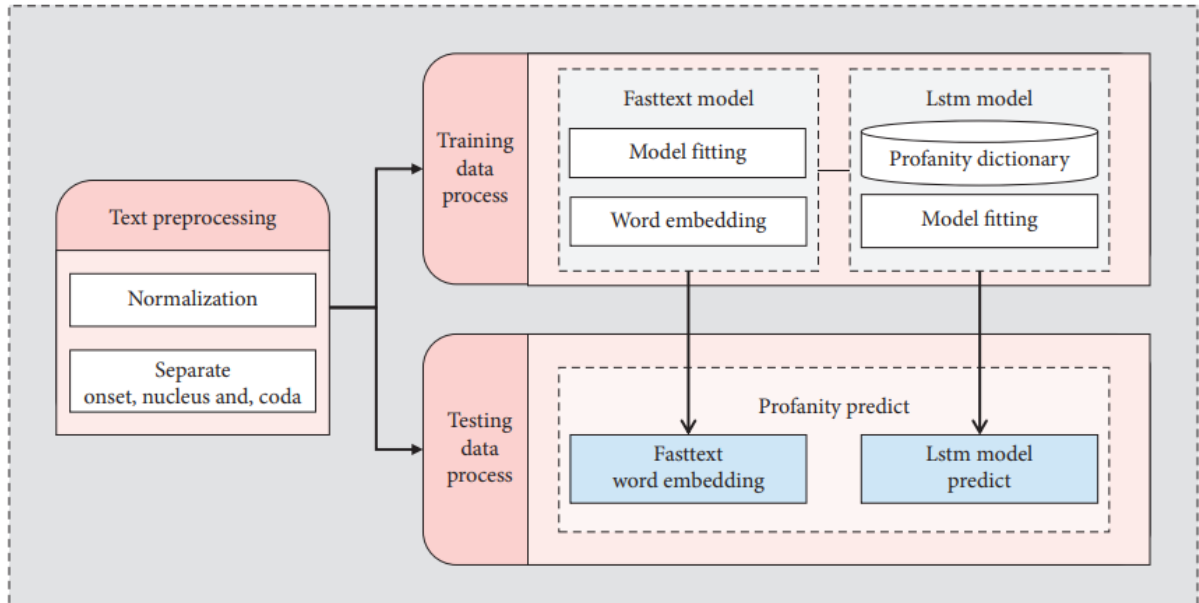


Figure 2-3: Overview of the system architecture[11].

Training Data Process involves text pre-processing, FastText and LSTM model training. In text preprocessing, the texts collected are normalized, and each word is separated into onset, nucleus, and coda. Then, the FastText model is trained with the separated text data, and word embedding will be performed on the text data using the vector information from the trained FastText model. In the LSTM model learning step, supervised learning for binary classification is performed. While training the LSTM model, the presence of profanity is checked, '1' represents profanity while '0' not.

For Testing Data Process, it requires the input to have the same format as the training process. Therefore, the same text preprocessing steps are also executed. Then, it will go through the word embedding step, then the existence of profanity is predicted as '1' or '0' using the LSTM model.

Text preprocessing includes 4 steps to be performed on the text data. First, special symbols and emoticons are removed. Second, one-syllable word is removed. Third, sentences with less than 5 words are removed. Fourth, each word is separated into onset, nucleus, and coda. Then, word embedding using the FastText model is performed with skip-gram at a learning rate of 0.05, dimension of 100, windows size of 5, 50 epochs, and n-gram of 1-6.

To demonstrate the effectiveness of the proposed profanity detection method, the accuracy, recall, and precision were measured. The data set used is collected from Twitter, with

1.3 million posts after duplicates were removed, and also 500,000 movie reviews collected from Naver. The ratio of the number of sentences with profanity and without is 1:1.

Figure 2-4 shows the analysis result. The proposed method (LSTM) is compared with CNN and GRU. For the CNN model, it has considerably lower performance when compared to other models with an accuracy of only 53%, recall of 13%, and precision of 79%. For the GRU model, the accuracy is 94.74%, recall is 95.18%, and precision is 94.32%. GRU model demonstrated a great performance, but it is still slightly less performant compared to the LSTM model. The LSTM model proposed in this study recorded an accuracy of 96.15%, which shows the highest performance among all compared methods. The cross-verification score for the LSTM model is 96%.

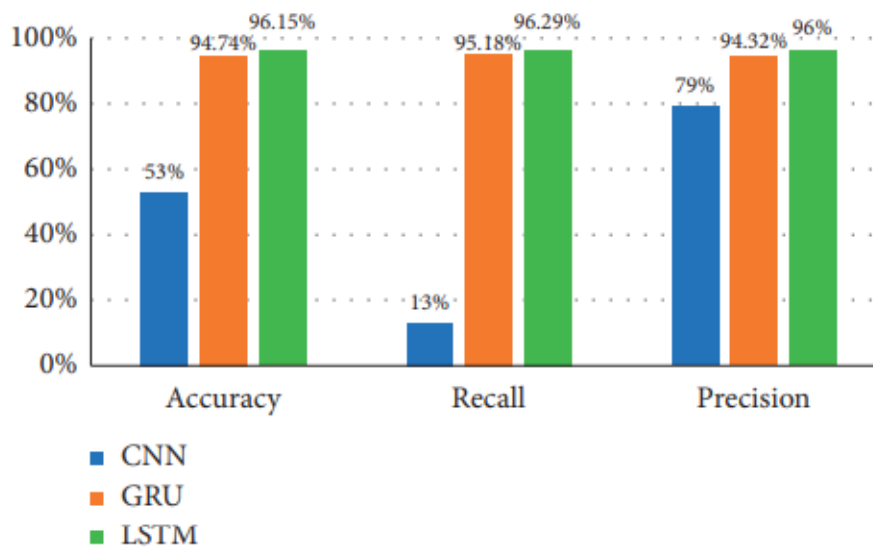


Figure 2-4: CNN, GRU, and LSTM model results [11].

To conclude, by using the FastText model, the method proposed by Yi, Lim, Ko and Shin, is able to capture the semantics and also the context used for the words. Furthermore, the flow context is trained by the LSTM, so it can recognize profanity that cannot be identified by other approaches. Based on the performance test result, the proposed method indicates great performance with an accuracy of 96.15% [11].

2.1.3 Automated Hate Speech Detection and the Problem of Offensive Language

A huge challenge for hate speech detection is to distinguish between hate speech from other types of offensive use of language. People often use words that are offensive to certain groups in a qualitatively different manner. For instance, some African Americans would use

'n*gga' as a normal phrase online. Therefore, to build a usable hate speech detection system, this boundary condition is critical.

Davidson, Warmesley, Macy and Weber, published a paper to resolve the challenges mentioned above. In the paper, they labelled Twitter tweets into 3 categories, which are hate speech, offensive language, and neither. A model is trained to classify messages within these classes, and results are analyzed. The results showed that classifying messages as these 3 classes can help boost the performance for the hate speech detection task.

They compared a variety of models like logistic regression, linear SVM, Naive Bayes, decision trees, and more. Every model is tested using 5-fold cross validation, with a training set to test set ratio of 0.9:0.1. After performing grid-search over the models and parameters, Logistic Regression and Linear SVM show better performance compared to other models. Thus, Logistic Regression with L2 regularization is selected as the final model as the predicted probabilities of class can be examined more easily.

The final model is trained using the entire dataset. The best performing model recorded a precision of 0.91, recall of 0.9, and F1 score of 0.9. Figure 2-5 illustrates the confusion matrix of the model. From Figure 2-5, almost 40% of hate speech is wrongly classified, for the hate class, the precision is 0.44 and recall scores is 0.61. This suggests the model is biased towards identifying Twitter's tweets as less offensive than the human.

Due to an overly broad definition, one of the major flaws in most of the previous works is the labelling, where offensive language is wrongly labeled as hate speech. The multi-class framework proposed by Davidson, Warmesley, Macy and Weber, minimizes these errors. Only 5% of their true offensive language was labeled as hate.

True categories	Hate	0.61	0.31	0.09
	Offensive	0.05	0.91	0.04
	Neither	0.02	0.03	0.95
		Hate	Offensive	Neither
		Predicted categories		

Figure 2-5: True versus predicted categories [12].

In conclusion, it is not that easy for a computer to differentiate between a perfectly fine speech and a hate speech. Many messages will be wrongly considered as hate speech if the meaning of hate speech and offensive language are conflated. While the results show that the model perform great at capturing some of the hate speeches, we should still be aware of the social biases that enter into the algorithms and these biases should be resolved in future work [12].

2.2 Existing Profanity Detection System

2.2.1 Profanity-check

Profanity-check is a robust Python library for checking profanity or offensive language in strings. It is an open-source project developed by Victor Zhou, and the source code is publicly available on GitHub.

Profanity-check provides 2 functions, `predict()` and `predict_prob()`, which can be used to detect profanity. Figure 2-6 shows the example usage of the functions and the output at the bottom of the figure. The function, `'predict()'` returns `'1'` if the string is profane while `'0'` if it is not. Meanwhile, `'predict_prob()'` returns the probability of profanity. Based on the output in Figure 2-6, it's clearly shown that profanity-check is capable of detecting profanity accurately. Profanity-check successfully determines `'hello everyone'` as not profane, `'fuck you'` as profane,

and surprisingly ‘your mom is fat cow’ which has only permissible words is also determined as profane.

```

from profanity_check import predict, predict_prob

print(predict([
    'hello everyone',
    'fuck you',
    'your mom is fat cow'
]))

print(predict_prob([
    'hello everyone',
    'fuck you',
    'your mom is fat cow'
]))

[0 1 1]
[0.03219666 1.          0.96422627]

```

Figure 2-6: Usage of profanity-check with Jupyter Notebook.

Profanity-check uses machine learning to perform predictions. The dataset used is combined from 2 sources. First one is Twitter-based dataset from Automated Hate Speech Detection and the Problem of Offensive Language by Tom Davidson, and second one is Wikipedia-based dataset from Toxic Comment Classification Challenge on Kaggle, published by Alphabet’s Conversation AI team.

After the collected text dataset is preprocessed, it will be passed to the vectorization process. In this case, Bag of Words (BoW) representation is used for vectorization. Basically, it turns a string into a numeric vector by counting the number of times each word appears in the strings. For example, one of the possible vectorizations results for ‘the ball drops into the hole’ is shown in Table 2-2. Finally, the vectorized dataset will be used to train the model. The model implemented in profanity-check is Linear Support Vector Machine (SVM), which is capable of performing different kinds of profanity detection in real-time.

the	ball	drops	into	hole
2	1	1	1	1

Table 2-2: BoW representation for ‘the ball drops into the hole’.

Based on the performance indicators published by Victor Zhou himself, profanity-check has high performance with an accuracy of 95%, precision of 86.1%, recall of 89.9%, and

F1 Score of 0.88. Although profanity-check is powerful, it is actually far from perfect as demonstrated in Figure 2-7. It fails to detect profanity with alternate spellings like ‘f4ck’. Moreover, it will determine a message as profane just because it contains the sensitive word, regardless of the context.

```

from profanity_check import predict, predict_prob

print(predict([
    'f4ck you', # profanity using alternate spellings
    'i suck my thumb' # not profanity with sensitive word 'suck'
]))

[0 1]

```

Figure 2-7: Failed profanity and non-profanity classification by profanity-check.

2.3 Critical Remarks of Previous Studies and Existing System

All of the previous studies reviewed have their own strengths and weaknesses. For the first study, it's capable of detecting profanity in multiple languages by using state-of-the-art cross-lingual transformer models, XLM-R. Meanwhile, the second study focuses on identifying profanity with alternating spellings. By using word embedding and LSTM, they proposed a model capable of identifying profanity even when the nucleus of the word is modified. The accuracy of the model is 96.15%. For the third paper, the proposed model can differentiate between hate speech and offensive language, which is a huge advantage.

For the system reviewed, which is profanity-check, it has its own strengths and weaknesses too. Profanity-check used a machine learning approach instead of list-based approach to implement profanity detection. This highly increases the accuracy of the system as there is no need for a hardcoded list to detect profanity. However, it is still unable to tackle alternating profane words like ‘f4ck’. Moreover, it is also unable to understand the context of the words used in the messages.

Based on the previous studies and existing system, a model to identify profanity for the text messages is proposed. By combining techniques from multiple studies and systems, the proposed model is capable of supporting multiple languages, understanding the context of words, and recognizing alternating spellings of profanity in text messages.

CHAPTER 3: METHODOLOGY

To develop the profanity detection API, various technologies and methodologies are applied. In this chapter, the methodology is first discussed. Then, the hardware and software requirements, and services used in this project are summarized.

3.1 Methodologies

In overall, the methodologies can be viewed as 2 different parts. The first part is the research phase which involves the system model, which the objective is to produce a profanity detection model. After the model is ready, the development phase which is the second part will focus on deploying the model as tools like API for users to use.

3.1.1 Dataset Acquisition

Since the profanity detection model proposed supports multiple languages, the dataset is required to be multilingual. For English, the dataset of Jigsaw Multilingual Toxic Comment Classification Challenge will be used. As for other languages, the dataset from Jigsaw Unintended Bias in Toxicity Classification Challenge will be used. Both of these datasets are available on Kaggle, and it is provided by the team from Conversation AI. With Kaggle CLI, these dataset can be downloaded into the notebook directly.

3.1.2 Model Training and Testing

This project proposes to use deep learning to detect profanity in the text messages, therefore a deep learning model will be designed and built. The model will be built by using various natural language processing techniques, combined with other techniques studied in literature review like cross-lingual emeddings, and LSTM. The dataset acquired will be split into training set and test set and then gone through preprocessing. Next, the training set will be used to train and validate the model, while the test set will be used to perform fine tuning and testing of the model. All of these methodologies will be done in Python programming language with Tensorflow library.

3.1.3 API Implementation

Once the model is completed and tested, it will be deployed as a REST API so that it can be implemented in other applications. The REST API for the proposed profanity detection

model will include two endpoints, 'is_profane' and 'profane_prob'. This will be implemented in Python using Flask-RESTful. Then, it will be deployed to localhost to perform testing on the system. Once final validation is completed, it is now ready to be deployed as production.

3.2 Tools

3.2.1 Hardware

The only hardware involved in this project is a computer. It will be used to develop the whole system. The computer used for this project is a personal laptop. The process of the development of the profanity detection model will be done using this personal laptop. Then, the REST API will be deployed on the localhost of this laptop for testing purposes.

Specification	Description
Model	Acer Aspire E1-572G
Processor	Intel Core i7-4500U 1.8GHz
Graphic	AMD Radeon HD 8750M 2GB VRAM
Memory	8GB DDR3 L Memory
Storage	1000GB HDD
Operating System	Windows 10

Table 3-1: Specification of laptop.

3.2.2 Software

Generally speaking, there are 3 software used to develop profanity detection API. The software is summarized as follows.

1. Visual Studio Code
Visual Studio Code is a code editor developed by Microsoft. It contains a lot of features like debugging, syntax highlighting and most importantly, a huge selection of extensions. In this project, VS Code is used together with the Python extension, and it will be used for the development, debugging, and testing of the RESTful API.
2. Keras
Keras is an open-source Python library that provides interfaces for artificial neural networks. It is built on top of the machine learning platform, Tensorflow. Keras

provides a high level of abstractions as well as being flexible enough to build custom models entirely from scratch. In this project, the building, training, and testing of the profanity detection model will be done using Keras.

3. Flask-RESTful

Flask-RESTful is an extension for Flask (micro web framework for Python), which adds support for building REST APIs easily. It is a lightweight abstraction that can work with existing libraries. Flask-RESTful will be used in this project to create REST API endpoints for the profanity detection model.

3.2.3 Services

Apart from hardware and software, this project also utilizes various services to aid the development and realization of this project. Some of the main services used are listed as follows.

1. Google Colab
Google Colab allows users to write and execute Python code through the browser in the form of the Jupyter notebook. The notebook environment will be run entirely in the cloud provided by Google. It offers free and paid plans. In this project, the free one is opted. Google Colab will be used as a platform to build, train, and test the model, as well as build a pipeline from dataset collection, preprocessing, to the output of the trained model. Training of the model is performed on Google Colab instead of the local machine because the computation power provided by Google Colab is far more powerful, it can speed up the computation time by a lot.
2. Heroku
Heroku is a cloud platform as a service (Paas) that supports several programming languages. Heroku provides hosted isolated Linux containers called dynos. It can be used to deploy and host web applications which are publicly accessible by everyone on the Internet. Apart from that, Heroku also provides managed SQL databases as a service called Heroku Postgres. In this project, free tier of the Heroku Dyno and Heroku Postgres are used.
3. Vultr
Vultr is a cloud hosting provider which provides various types of servers including

VPS, and also dedicated servers. It can be used to host any kind of application. It provides SSH access to the server, therefore the server can be set up easily. In this project, a paid service of Vultr is used, which is called Cloud Compute. It is used to perform the calculation profanity detection of the deep learning model on the cloud.

3.3 Timeline

The development of the whole project is split into 2 parts, namely Project 1 and Project 2. Project 1 completed 30% of the work. The aim of Project 1 is to build an overall architecture of the system as a proof of concepts. A simple deep learning model for profanity detection will be trained and deployed in the form of REST API locally. Meanwhile, Project 2 focuses more on the studies of the deep learning model, and designing, improving the profanity detection model so that it meets the objectives of understanding the context of bad words, alternative spellings, and multiple languages. The following figures show the timeline for both Project 1 and 2.

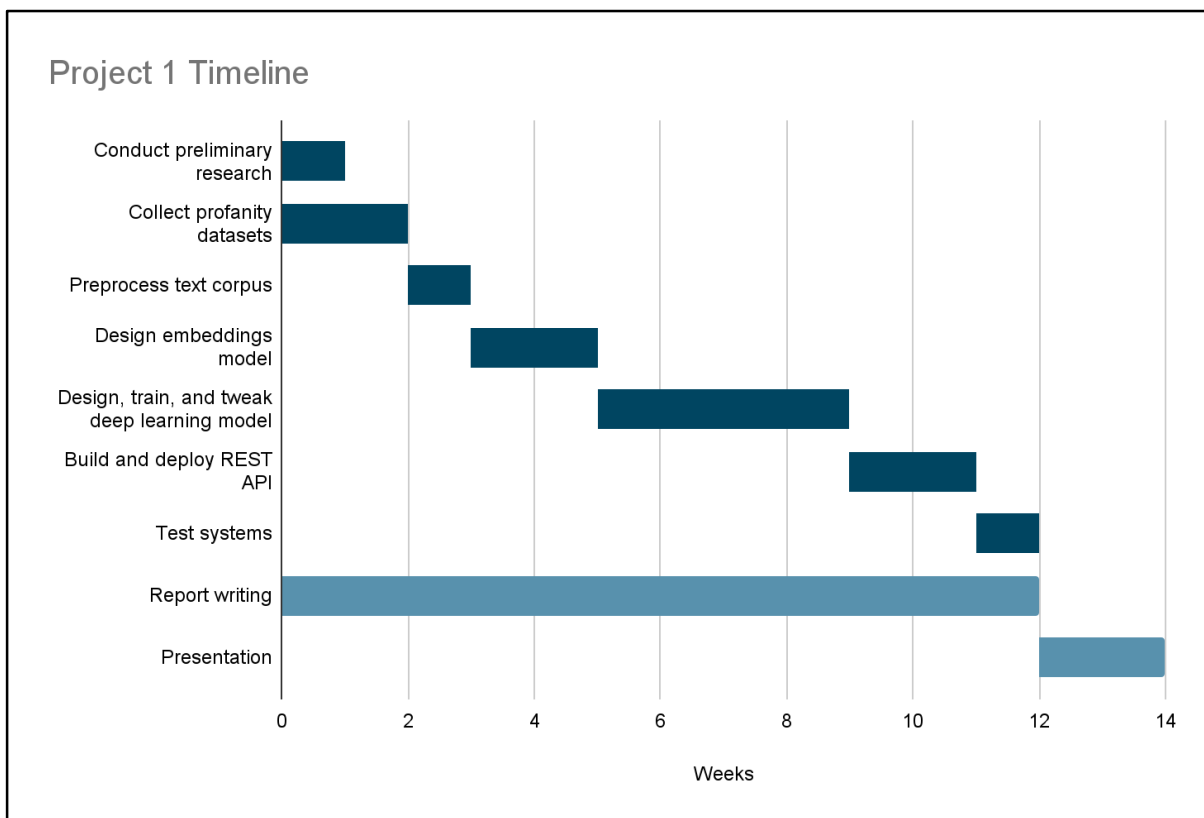


Figure 3-1: Gantt chart for the timeline of project 1.

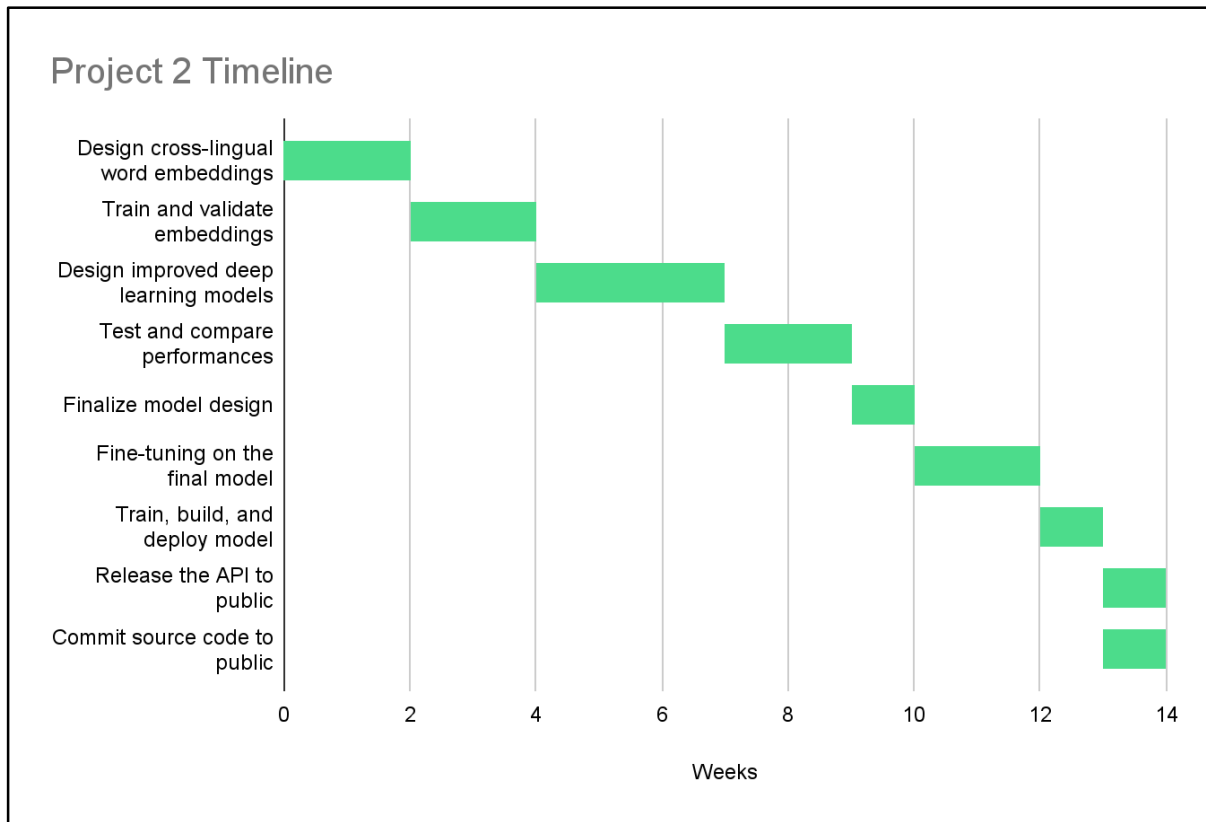


Figure 3-2: Gantt chart for the timeline of project 2.

CHAPTER 4: SYSTEM MODEL

As discussed in this report previously, the model used in this project to detect profanity is a deep learning-based model instead of a traditional dictionary-based approach. There are 2 different models proposed in this project, one is LSTM, another one is BERT.

4.1 LSTM with FastText Embeddings

Therefore, a deep learning binary classification model is designed as follows. Conceptually, the model utilizes bidirectional Long Short Term Memory (LSTM) to perform feature extraction. Then, the features extracted from LSTM will be passed to the fully connected layer to perform prediction.

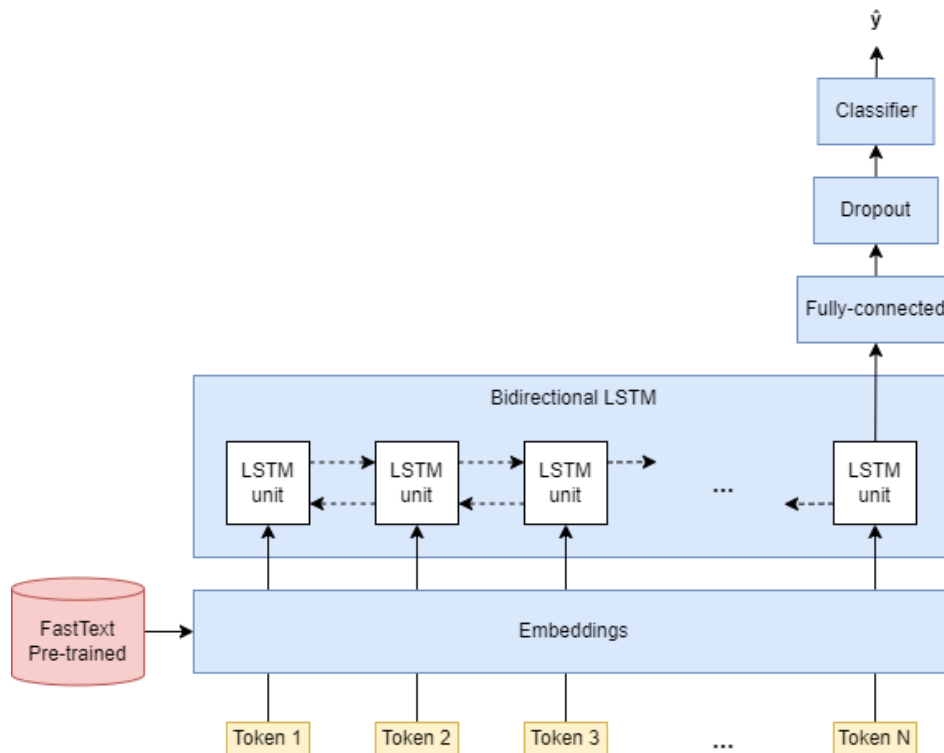


Figure 4-1: Architecture of the LSTM profanity detection model.

As shown in Figure 3-1, the model has a scalar output, \hat{y} , ranging from 0 to 1, which can be depicted as the probability of profanity for the given text message. Next, the model does not accept the whole sentence directly as the input. The text message has to be first tokenized into N tokens. Each token can be used to represent a single word in the text message. The following table shows an example of the tokenization process.

"Hi, how are you?"			
<i>Token 1</i>	<i>Token 2</i>	<i>Token 3</i>	<i>Token 4</i>
hi	how	are	you

Table 4-1: Example tokenization process.



Figure 4-2: Example word embeddings.

The first layer of the model is the embedding layer. This layer is responsible for converting each word token into a vector which can represent the meaning of the word. It works by referring to an embedding matrix as shown in Figure 3-2, for instance. The word embeddings is designed in such a way that the words that have vectors close to each other are expected to have similar meaning. Therefore, word embeddings can be leveraged to bring the semantic meaning of the word into the deep learning model, enabling the model to perform even better. For example, as displayed in Figure 3-2, “male”, and “female” are mapped into [1, 5, 0, 3], and [-1, 5, 0, 3], respectively. Notice that the vectors for both of these words are pretty close to each other, this suggests that “male”, and “female” have similar meanings, in this case, both of the words are the type of gender. In contrast, “zebra” which is an animal type has a very different vector.

There are quite a few word embedding techniques proposed recently, for example, Google’s Word2Vec, GloVe, and also Facebook’s FastText. The embedding matrix can be produced from scratch by training on the datasets collected or downloaded from the pre-trained embedding model online. In this project, FastText’s English pre-trained model is used. FastText is selected over other techniques due to the fact that FastText improves Word2Vec by operating at a more granular level with character n-grams, taking part of the words and characters into account. This allows training of embeddings on smaller datasets and rare words

to be represented appropriately. This is important for profanity classification tasks as there are all kinds of less frequently used words that can appear in a profane message. Apart from that, the pre-trained model is used because it is trained on large datasets with specifications from the team of the researchers themselves, so it generally performs better for most of the tasks.

After the tokens are converted into vectors, it can now be passed to the next layer, the bidirectional LSTM. The goal of this layer is to extract features from the text message so that the model can be trained and learn from the features provided. The LSTM layer is constructed from the LSTM cell.

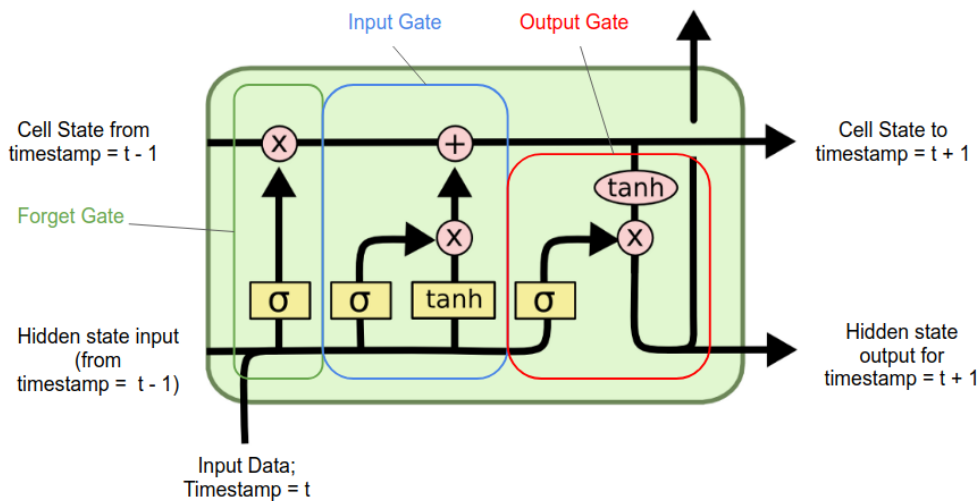


Figure 4-3: A single LSTM Cell [13].

In a LSTM cell or unit, it is made up of several components, the cell state, the hidden state, the forget gate, input gate, and output gate. The hidden state is responsible for encoding the information of the data at the given time-step. The information of the hidden state depends on what the LSTM is asked to do (trained), the weights will be updated accordingly during the training phase. In this project, the hidden state can be expected to represent information about if this word carries any negative sentiment. As for the cell state, it acts like a memory cell, trying to encode all the information in the previous time-steps. Notice that the cell state is connected with the forget gate, input gate. This means its content can be kept, erased, and updated. This feature introduces complexity to the model so that it is capable of capturing the important feature in the profane message.

Next, the output of the LSTM cell at the last time-step will be taken for the classification process. The information encoded at the last time-step is expected to represent the features for the whole sentence. This feature will be passed into a fully connected layer with output size of 32, and ReLU activation function. This fully connected layer is responsible for learning the pattern of profanity from the features provided. Then, Dropout is applied after this fully connected layer. Dropout layer randomly dropping out input units with a specified frequency at each step during the training phase. By doing so, it can help prevent overfitting of the model. For this project, the dropping frequency for the Dropout layer is set to 0.4. Next, the features will pass to the classifier. The classifier is just a fully connected layer at its core, but with only 1 output unit with sigmoid activation function. Sigmoid function squishes any input into the range of 0 to 1, therefore the output of this classifier layer can be treated as the probability of the message being profane. Finally, a threshold can be set to decide the minimum probability required for a message to be considered as profane. For example, a threshold of 0.5 will mark all the messages with probability of 0.5 and above as profane.

4.2 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers, which is also known as BERT, was developed by Google back in 2018 [14]. There are 2 variants of BERT presented in the paper. The first one is called BERT Base, and another one is called BERT Large. BERT Large provides state of the art performance, but it is not preferable in this project, as the training time of BERT Large is long. BERT Base should achieve a satisfactory result for profanity detection tasks. BERT Base contains 12 Transformer Blocks, feedforward-networks with 768 hidden units, and 16 attention heads.

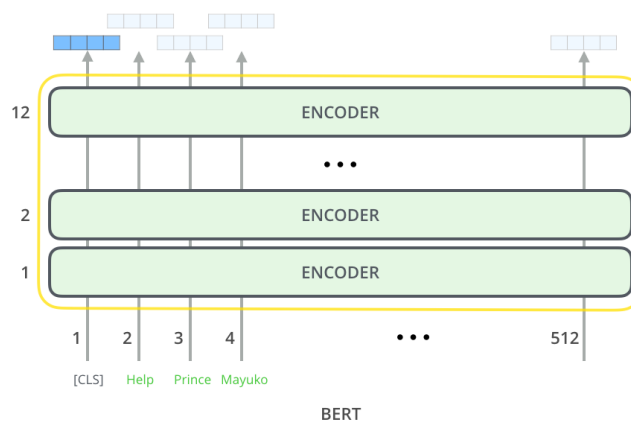


Figure 4-4: Input tokens and output tokens of BERT.

Similar to LSTM, text messages have to be tokenized before input to BERT. However, there is a special token in BERT called [CLS], which stands for Classification. This [CLS] token will be supplied into BERT as the first token. Since the number of hidden units for the feedforward-networks in BERT base is 768, therefore the token at each position is a vector of size 768. As shown in Figure 4-4, the output tokens are vectors. The output vector at the first position can be treated as the encoding of the meaning of the whole sentence, as the input at the first position is supplied with the special [CLS] token. This means this vector can be used to perform classification tasks.

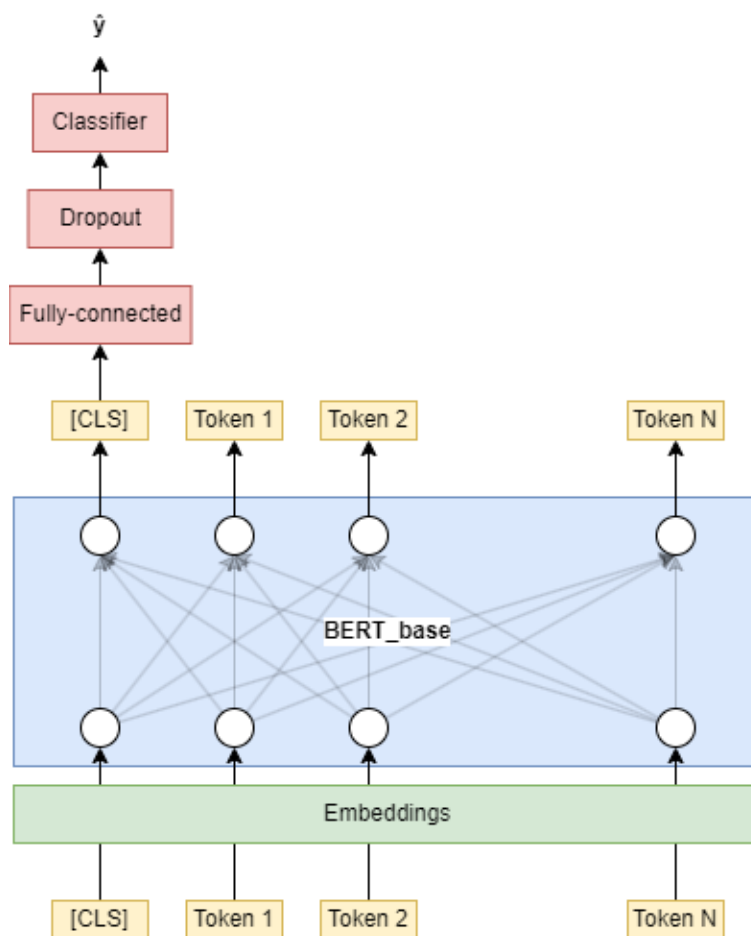


Figure 4-5: Architecture of profanity detection model based on BERT.

Figure 4-5 shows the model design of BERT for profanity detection tasks. The BERT encoder and the embedding layer used is pre-trained. The pre-trained model is open-sourced and provided by the original author of BERT, it is trained with the text data from Wikipedia and books. The pre-trained model is capable of understanding the language, like the grammar. There is no need to train BERT from scratch as it will take a huge amount of time and usually

the end-result is worse than using pre-trained. BERT is based on Transformer, and Transformer has a Embedding layer which includes positional embedding. There is no need to use an external embedding like FastText. The text input can be directly passed into the Embedding layer provided, and it will turn the text into tokens which are essentially embeddings. Only then, these tokens will be fed into the BERT encoder layer.

The output of the BERT encoder at the first position is taken for the classification of profanity. This output vector is passed into a fully connected layer with an input size of 768 and output size of 512. Then, the result is fed into a Dropout layer with the dropping frequency set as 0.1. Next, the features will pass to the final layer which is the classifier. The classifier is another fully connected layer with output size of 1. The activation function used in the classifier is sigmoid, which means the final predicted output will range from 0 to 1.

CHAPTER 5: SYSTEM DESIGN

This chapter documented the development phase of the project. The deliverable of the development phase is a package called DeepProfane with a suite of tools for the users to filter profane messages effectively. By making use of the deep learning model developed from the previous chapters, DeepProfane is capable of recognizing profanity in the text messages and then leveraging it to build profanity detection API and software. The tools developed include a PyPI Python package (deep-profane), a RESTful API (deep-profane-rest), and a web chatting application capable of filtering profane messages automatically (DeepProfane Chat).

5.1 System Block Diagram

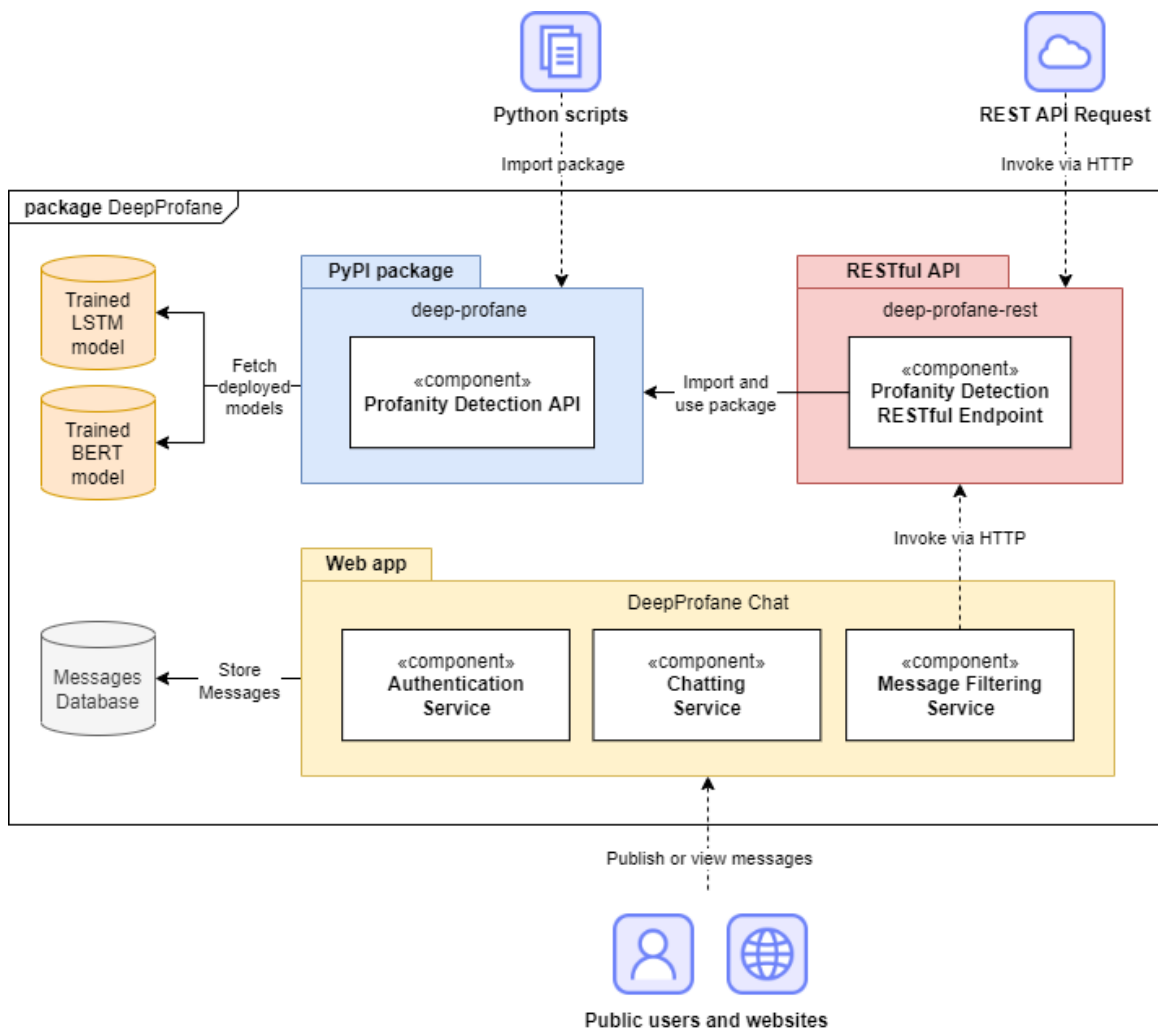


Figure 5-1: Block diagram of the DeepProfane package, which has 3 modules, including PyPI package, RESTful API, and a web app.

As shown in the block diagram above, the final deliverable of this project is a package called DeepProfane which contains several modules. Each module is designed as an individual function that can be used publicly by end users or internally by other modules.

The first module in this package is the **deep-profane** module. The deep-profane module is published as a PyPI package, and it is publicly accessible by any Python program. It performs profanity detection by employing the trained LSTM model and BERT model from the previous chapter. It is the most fundamental and important module as it is the only module in the package to power the recognition of profanity. Based on Figure 5-1, notice that all profanity detection tasks in this package will be routed to this module. Since this is a Python package, this module can be directly downloaded and imported from PyPI, and then integrated into Python scripts to perform profanity detection.

Apart from that, this package also includes a profanity detection REST API called **deep-profane-rest**, which is publicly available to anyone. Unlike the deep-profane module which can only be used in Python scripts, deep-profane-rest is accessible via HTTP protocol. This means deep-profane-rest is more flexible and can be used by programming languages other than Python. At the core of deep-profane-rest, it performs profanity recognition by using deep-profane, the classification result retrieved from the deep-profane module is routed back to its user. In overview, deep-profane-rest encapsulates deep-profane and it is a gateway for its users to access the profanity detection API via HTTP.

The last and the most complex module is **DeepProfane Chat**. It is a web chatting application with profanity filtering enabled. DeepProfane Chat can be embedded into custom websites to replace the conventional commenting and chatting system. Public users can publish their messages on DeepProfane Chat, and the messages will be stored at a database. Every message will first be passed to profanity detection service to filter out any profanity. All messages detected with profanity will be flagged and will not be displayed to other users. This automated moderation process is achieved by using deep-profane-rest, which uses deep-profane at the end.

5.2 PyPI Package (deep-profane)

This section describes the process of developing and packaging the profanity detection model as a Python package. The name of the package is called deep-profane.

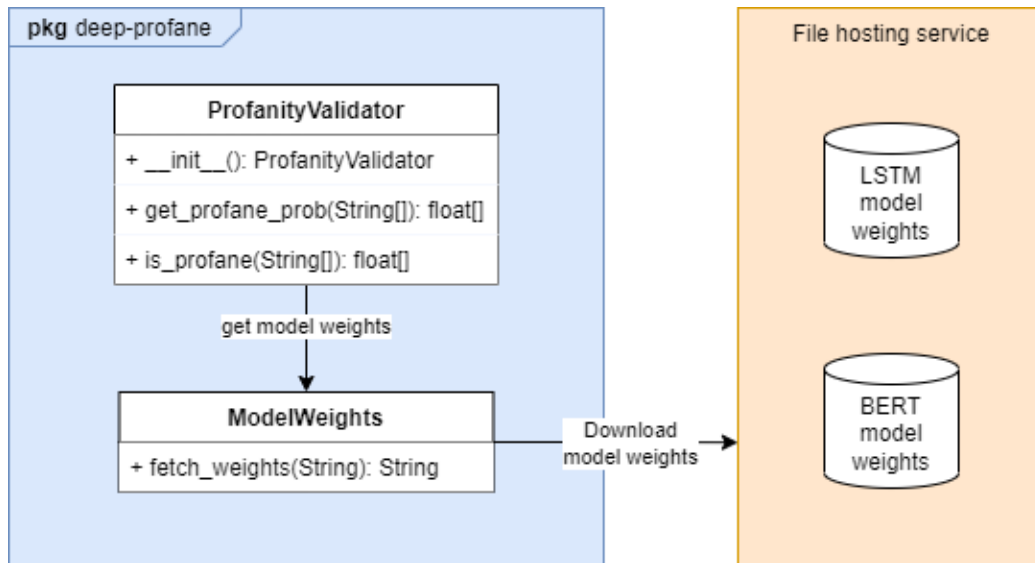


Figure 5-2: Overview class diagram for deep-profane package.

First of all, the profanity detection model developed is not distributed together with deep-profane as a single Python package. Instead, the model weights are hosted on a third party file hosting service. The model will be downloaded automatically when the user installs and runs deep-profane on their machine. This is because the file size for model weights is too large, exceeding the package size limit imposed by PyPI, which is 60MB [15]. Usually, the file size for a single BERT model is more than 100 MB.

There are 2 main classes in the deep-profane package. The first one is ProfanityValidator. ProfanityValidator is the main interface for users to invoke and perform profanity detection, it exposes 2 functions for this usage. The first one is a function to get the probability of profanity called get_profane_prob(). This function accepts a list of strings and outputs a list of probabilities ranging from 0 to 1. Similar to the get_profane_prob() function, the second function called is_profane() returns a list of booleans to indicate if the given list of strings is profane or not.

The get_profane_prob() and is_profane() function calculate the profanity result by using the model constructed and loaded by ProfanityValidator. In the initialization function, __init__(), of the ProfanityValidator, it will construct a model, and store it in the memory. Then, the function will call fetch_weights in ModelWeights to download weight data from the sources and then load it into the newly constructed model. Only at this point of time, the package is ready to be used.

5.3 RESTful API (deep-profane-rest)

The second module is called deep-profane-rest, it is a RESTful API for profanity detection.

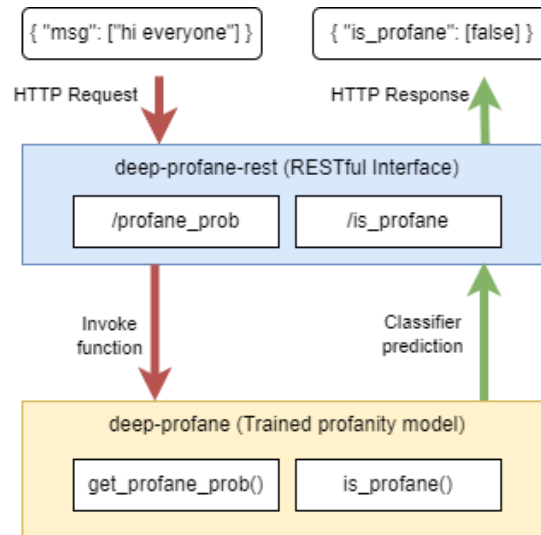


Figure 5-3: Architecture for deep-profane-rest, profanity detection RESTful API,

Request	Response (isProfane endpoint)	Response (probability endpoint)
<pre>{ "msg": ["good morning", "fuck you", "i suck my thumb"] }</pre>	<pre>{ "is_profane": [false, true, false] }</pre>	<pre>{ "profane_prob": [0.3785, 1, 0.4012] }</pre>

Table 5-1: Sample request and responses of Profanity Detection REST API.

The deep-profane-rest module serves as the gateway for third-party websites or applications to use the profanity detection library, deep-profane. The RESTful architecture is chosen because it provides a uniform and standard way of communication between the server and the client. This means the invocation of this API is not constrained to a particular programming language. Any programming language can use this API as long as the API request follows the specification of the RESTful API endpoint. As shown in Figure 4-5, the request will first go through the REST Interface layer, then routed to the classification model after it is pre-processed. The detection result will then pass back to the users through the REST Interface.

There are 2 endpoints for the deep-profanest, which are `/is_profanest`, and `/profane_prob`. The endpoint `/is_profanest`, returns boolean values (true or false) to indicate if the message is profane or not. As for `/profane_prob`, it returns the probability of the message being profane. As displayed in Table 4-1, the API is designed to communicate in JSON format. For each request, users can send up to 8 messages. The messages must be contained in a list with `msg` as the key. Otherwise, the request is considered as invalid, and an error will be thrown. This is to limit the workload of the REST API server from growing too large, making it impossible to handle other requests in the case where a single request sends an enormous number of messages. The same constraint applies to the profane_prob endpoint as well.

5.4 Web Chatting Application (DeepProfane Chat)

This section will describe the design of DeepProfane Chat module.

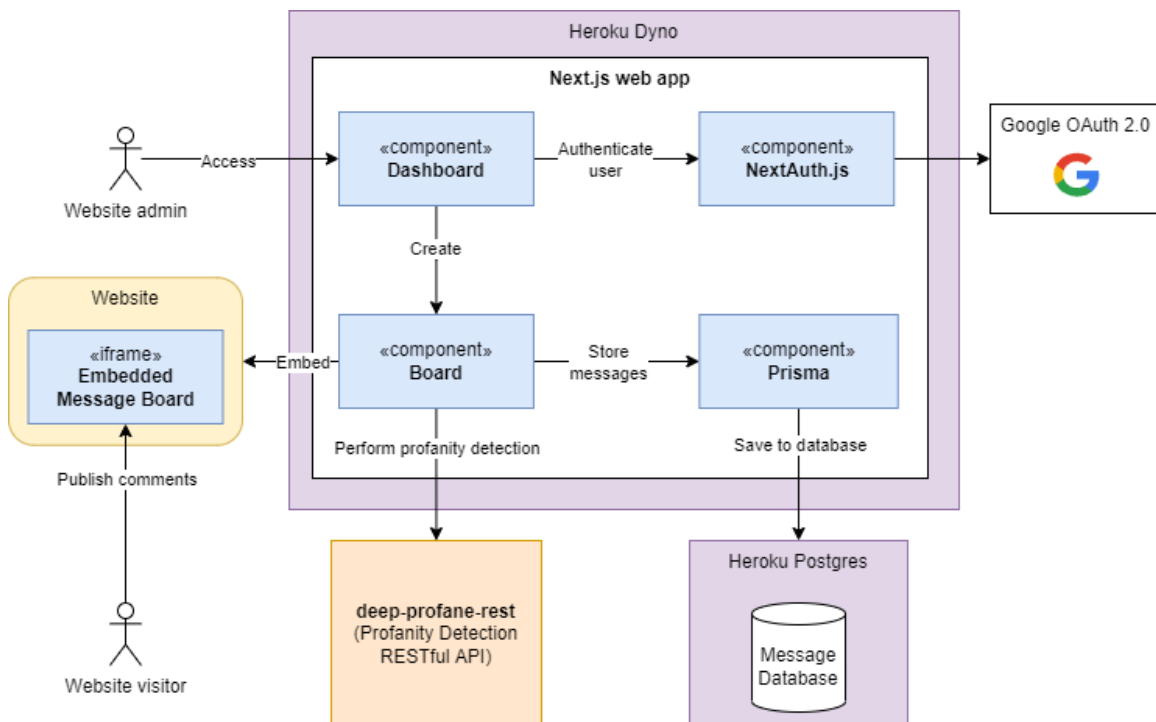


Figure 5-4: Simplified block diagram for DeepProfane Chat module.

The key feature of DeepProfane Chat is it is designed as a third-party website embeddable chatting system, but not as a standalone chatting system. This means that the website of DeepProfane Chat does not provide any chatting service, but instead the chatting service can be integrated into various websites. The intention of this design is to replace the conventional chatting or commenting system on the website in which there is usually no

profanity moderation. The embeddable chatting service provided by DeepProfane Chat has profanity detection enabled and it requires minimal setup by website owners. This requirement is achieved by using the iframe technology as shown in the yellow block inside Figure 5-4. By using iframe, DeepProfane Chat will handle the render of the chat interface called 'Board', and then the iframe setup at the website can integrate this Board into the website by specifying the 'src' properties (URL) of the iframe.

Referring to Figure 5-4, the stack used in DeepProfane Chat is Next.js. All components inside the Next.js block will be integrated in the Next.js stack. The first entry point of this web application is the Dashboard. The users or website owners can access the Dashboard to perform setup and integration of DeepProfane Chat into their website. The dashboard is a protected component which means the users have to authenticate themselves before they can access the dashboard. The authentication system is implemented by using OAuth 2.0 protocol with NextAuth.js and Google OAuth 2.0 as the provider. This means DeepProfane Chat does not handle and store the username and password of the accounts, instead it is provided by Google. The users can simply login to the dashboard by using their Google account.

The dashboard allows website owners to perform various actions. First, they can create a new Board for their website. Then, they can view the configuration and API Key of this Board, in which they can copy paste the iframe code generated at the dashboard into their website to integrate the Board. Also, they can view the number of messages posted, and the number of messages blocked for profanity at the dashboard.

A Board is the chatting interface that can be embedded into a third party website. The visitor of the website can read or publish comments on the Board setup by the website. These messages or comments will automatically be sent to deep-profane-rest for profanity screening. If profanity is detected, the comments will be flagged and will not be shown to the users. However, the website owners can view these flagged messages at the dashboard. All messages will be stored in a database via an ORM library called Prisma.

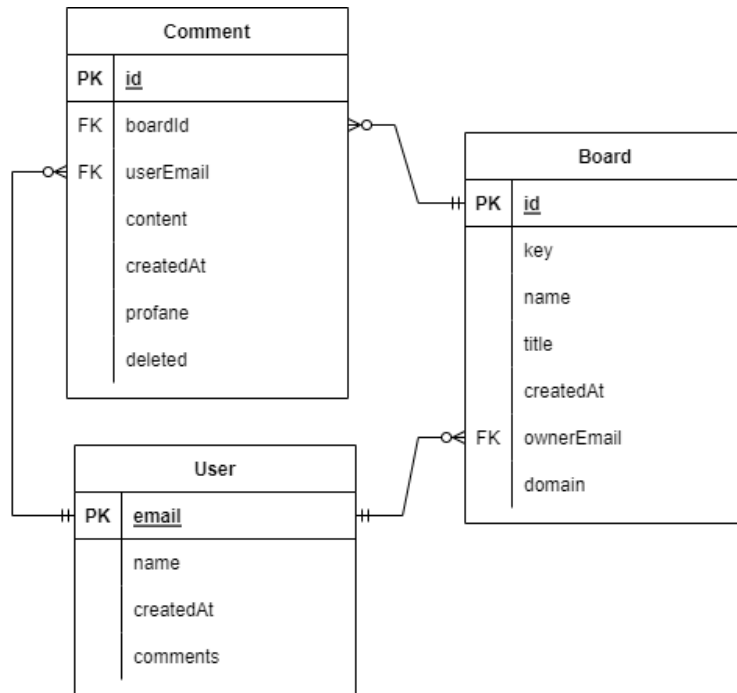


Figure 5-5: Entity relation diagram for the database of DeepProfane Chat.

The schema of the database is shown in Figure 5-5. There are a total of 3 tables. The first table is called User. The User table holds information about the website owner or website visitor who published comments at DeepProfane Chat. Each user is allowed to create more than one Board. The Board table contains information about the Board like its name, its domain, API key, the owner of the Board, etc. Each Board can have more than one Comment associated. In the Comment table, the column 'content' holds the comment text itself, and 'profane' is a boolean column which can indicate if this comment is flagged for profanity. Each comment must be associated with one User and one Board. This means that the users have to login into an account before they can publish comments on DeepProfane Chat.

The database and the web application itself will be hosted on Heroku Dyno and Heroku Postgres, respectively. This will make it accessible from the public, and its implementation details will be discussed in the next chapter.

CHAPTER 6: SYSTEM IMPLEMENTATION

In this chapter, the implementation details of the profanity detection system is documented. In short, there are 2 sections in implementation of the system. The first section is about the experiments of the profanity detection deep learning model discussed in Chapter 3. The second section is the development of the DeepProfane package as discussed in Chapter 4.

6.1 Software Setup

First of all, prior to the implementation of the system, the development environment is required to be set up by installing these software on the local machine:

1. Visual Studio Code
2. Python extension for Visual Studio Code
3. Python 3.9.12
4. PyPI CLI
5. Postman
6. Heroku CLI

6.2 Service Setup

Then, a notebook document (.ipynb) is created and hosted on Google Colab, with GPU runtime enabled. This document will be used to build, train, and test the model. Apart from that, a Heroku account, a PyPI account, and a GitHub account are also required to be registered.

6.3 Implementation of Profanity Detection Model

Based on the model design as discussed in Chapter 4, there are 2 models needed to be built and tested. The first model is LSTM-based with FastText embeddings, and another one is BERT model. Both of these models will be configured and trained with the notebook document (.ipynb) created earlier which is hosted on Google Colab. The model with best performance will be chosen as the final model.

6.3.1 Datasets Acquisition

In order to build and train the profanity detection model, datasets relevant to the abusive comments have to be collected. In this project, 2 different datasets are used:

1. Dataset of Jigsaw Multilingual Toxic Comment Classification by Conversation AI.
2. Dataset of Jigsaw Unintended Bias in Toxicity Classification by Conversation AI.

These 2 dataset collections have their own train set, validation set, test set, and other metadata are separated in different CSV files. However, not all of these CSV files are used in this project. Only the train set (jigsaw-unintended-bias-train.csv), and test set (test.csv + test_labels.csv) from Multilingual Toxic Comment Classification and also the test set (test.csv + test_labels.csv) from Jigsaw Unintended Bias in Toxicity Classification are used.

```
[ ] !kaggle competitions download -c jigsaw-multilingual-toxic-comment-classification

Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.7/dist-packages/kaggle/_init_.py", line 23, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.7/dist-packages/kaggle/api/kaggle_api_extended.py", line 166, in authenticate
    self.config_file, self.config_dir))
OSError: Could not find kaggle.json. Make sure it's located in /root/.kaggle. Or use the environment method.
```

```
[ ] # Extract files zip downloaded from Kaggle
with zipfile.ZipFile(u.PROJ_DIR + "/jigsaw-multilingual-toxic-comment-classification.zip", 'r') as zip_ref:
    zip_ref.extractall(u.JIGSAW_MULTILINGUAL_DS_DIR)

# Extract all .csv zip inside the files zip
for file in os.listdir(u.JIGSAW_MULTILINGUAL_DS_DIR):
    if zipfile.is_zipfile(u.JIGSAW_MULTILINGUAL_DS_DIR + "/" + file):
        with zipfile.ZipFile(u.JIGSAW_MULTILINGUAL_DS_DIR + "/" + file) as item:
            item.extractall(u.JIGSAW_MULTILINGUAL_DS_DIR)
```

Figure 6-1: Source code to download and unzip Jigsaw Multilingual Toxic Comment Classification dataset from Kaggle.

A pipeline to automate this process of downloading the datasets into the notebook document is written. Since both datasets are available on Kaggle, these datasets are downloaded using the Kaggle CLI tool, and then extracted using zipfile library in Python.

6.3.2 Datasets Exploration

	comment_text	toxic
0	This is so cool. It's like, 'would you want yo...	0.000000
1	Thank you!! This would make my life a lot less...	0.000000
2	This is such an urgent design problem; kudos t...	0.000000
3	Is this something I'll be able to install on m...	0.000000
4	haha you guys are a bunch of losers.	0.893617
...
1902189	He should lose his job for promoting mis-infor...	0.000000
1902190	"Thinning project is meant to lower fire dange...	0.166667
1902191	I hope you millennials are happy that you put ...	0.400000
1902192	I'm thinking Kellyanne Conway (a.k.a. The Trum...	0.000000
1902193	I still can't figure why a pizza in AK cost mo...	0.000000

Figure 6-2: Preview of the train set of Multilingual Toxic Comment Classification.

In the train set of Multilingual Toxic Comment Classification, there are a total 45 different columns, but the only columns relevant to the profanity detection task in this project are ‘comment_text’ and ‘toxic’. The column, ‘comment_text’ holds the text corpus, and the column ‘toxic’ specifies the probability of profanity in the range of 0 to 1. There are a total of 1,902,194 entries in this train set. Out of these 1,902,194 entries, 152,111 of them have toxicity of equals or greater than 0.5, which is considered as toxic or profane.

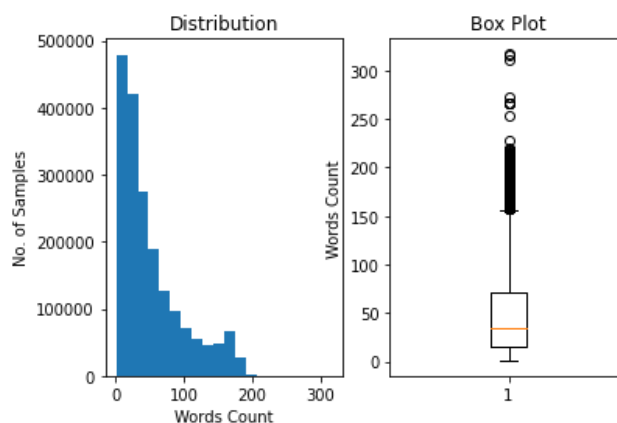


Figure 6-3: Distribution and box plot of words count for all text in the train set.

Since BERT is one of the models to be trained, it would be useful to gain insight about the number of words for the text corpus of each entry. This is because there is a hyperparameter

to specify the number of input tokens in the BERT model. Figure 6-2 shows the statistics of the word count distribution. Based on the distribution chart, most of the entries have a number of words less than 200. In fact, based on the analysis of boxplot and interquartile range, there are only 106,904 (5.62%) entries with a number of words of more than 156, which are considered as outliers. The implementation of BERT at the later section can select the input tokens to be 128-256 based on the overview provided here.

id	content	lang	toxic
0	Doctor Who adlı viki başlığına 12. doctor olar...	tr	0
1	Вполне возможно, но я пока не вижу необходимо...	ru	0
2	Quindi tu sei uno di quelli conservativi , ...	it	1
3	Malesef gerçekleştirilmedi ancak şöyle bir şey...	tr	0
4	:Resim:Seldabagcan.jpg resminde kaynak sorunu ...	tr	0
...
63807	No, non risponderò, come preannunciato. Prefer...	it	0
63808	Ciao, I tecnici della Wikimedia Foundation sta...	it	0
63809	innazitutto ti ringrazio per i ringraziamenti...	it	0
63810	Kaç olumlu oy gerekiyor? Şu an 7 oldu. Hayır...	tr	0
63811	Te pido disculpas. La verdad es que no me per...	es	0

Figure 6-4: Preview of the test set of Multilingual Toxic Comment Classification.

As shown in Figure 6-4, the text corpus in the test set consists of multiple languages other than English. There are a total of 3 columns in the test set, which are content, lang, and toxic. The column 'content' is the text corpus, meanwhile the column 'lang' recorded the language for the respective sentences, and the value for the column 'toxic' is either 0 or 1 which indicates non-toxic or toxic. The following table shows all the languages included in the text corpus of the test set, there are a total of 6 languages.

Code	Language
tr	Turkish
ru	Russian
it	Italian
fr	French
pt	Portuguese
es	Spanish

Table 6-1: Language code and its name for all the supported languages in the test set.

6.3.3 Training of LSTM Model

LSTM is a model with no word embeddings on its own. Therefore, the dataset which is in text (string) form cannot be fed into the LSTM model directly. Pre-processing on the text corpus in the dataset has to be done before proceeding to the training of the LSTM model.

```

from keras.preprocessing.text import Tokenizer

MAX_WORDS_COUNT = 120000

tokenizer = Tokenizer(num_words=MAX_WORDS_COUNT, filters=CHARS_TO_REMOVE)
tokenizer.fit_on_texts(list(x_train) + list(x_test))

x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)

word_index = tokenizer.word_index
print("dictionary size: ", len(word_index))

dictionary size: 613611

```

Figure 6-5: Source code for tokenizing the input text corpus in the notebook document.

First of all, the dataset is passed to a standard Tokenizer to map every word in the text corpus to a number called token. As displayed in Figure 6-5, this is done by using Tokenizer provided from keras library. Notice that the output says there are a total of 613,611 words in the dictionary, but the maximum allowed size of the dictionary is capped at 120,000 words. This means any word ranked outside of the top 120,000 words in the dictionary will not be mapped into its respective token.

```

def len_2d(arr):
    return np.array(list(map(lambda x: len(x), arr)))

lens = len_2d(x_train)
max_len = np.max(lens)

print(f'Before padding: min={np.min(lens)}, max={max_len}')
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
print(f'After padding: min={np.min(len_2d(x_train))}, max={np.max(len_2d(x_train))}')

Before padding: min=0, max=320
After padding: min=320, max=320

```

Figure 6-6: Source code for padding the tokenized text corpus.

After the tokenization process is completed, the length of the list tokens for every sentence will be different as each sentence has a different number of words. Therefore, padding is required to be performed on the list of tokens, so that every sentence in the train and test set have the same number of tokens. This makes it easier for the training process later in the LSTM model. Figure 6-6 shows the code to perform sequence padding. First, the maximum length in the whole text corpus is calculated and then every sentence will be padded according to this length. In the example shown in the Figure, the maximum length is 320.

```
zip_file_url = "https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl-300d-2M.vec.zip"
r = requests.get(zip_file_url)
z = zipfile.ZipFile(io.BytesIO(r.content))
z.extractall(path=f'{PROJ_DIR}/fasttext')

embeddings_index = {}
f = codecs.open(f'{PROJ_DIR}/fasttext/crawl-300d-2M.vec', encoding='utf-8')

for line in tqdm(f):
    values = line.rstrip().rsplit(' ')
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print(f'found {len(embeddings_index)} word vectors')

1999996it [02:31, 13185.08it/s]found 1999996 word vectors

words_not_found = []
words_count = min(MAX_WORDS_COUNT, len(word_index))

embedding_matrix = np.zeros((words_count, 300))

for word, i in word_index.items():
    if i >= words_count:
        continue
    embedding_vector = embeddings_index.get(word)
    if (embedding_vector is not None) and len(embedding_vector) > 0:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
    else:
        words_not_found.append(word)

print('shape of embedding matrix: ', embedding_matrix.shape)
print('number of words with embeddings not found: %d' % np.sum(np.sum(embedding_matrix, axis=1) == 0))
print("sample words not found: ", np.random.choice(words_not_found, 10))

shape of embedding matrix: (120000, 300)
number of words with embeddings not found: 29184
sample words not found: ['ненавижу' 'reversor' 'foval' 'эй' 'trollare' 'stubway' 'mitterrand'
'toronna' 'ussa' 'vapmachado']
```

Figure 6-7: Source code for downloading FastText embeddings and transforming the words (tokens) into embedding vectors.

Then, the last step of the pre-processing is converting the tokens into its embedding vector. FastText's pretrained English embeddings data is downloaded and used to achieve this task. As shown in Figure 6-7, all tokens in all sentences are converted to FastText's embeddings vector one by one by using a for loop. The pre-processing is completed once this conversion is finished.

```

model = tf.keras.Sequential()

model.add(Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False))
model.add(Bidirectional(LSTM(32)))
model.add(Dense(128,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.summary()

Model: "sequential_3"
-----
Layer (type)                 Output Shape              Param #
-----
embedding_2 (Embedding)      (None, None, 300)        36000000
bidirectional_2 (Bidirectio  (None, 64)                85248
nal)
dense_4 (Dense)              (None, 128)              8320
dense_5 (Dense)              (None, 1)                129
-----
Total params: 36,093,697
Trainable params: 93,697
Non-trainable params: 36,000,000

```

Figure 6-8: Source code for constructing the proposed LSTM profanity detection model.

The model is constructed according to the proposed LSTM model in Chapter 4 as shown in Figure 6-8. There are a total of 93,697 trainable parameters. Notice that the embedding layer has the trainable parameter set to false to prevent the training process from updating its weights. After that, the model is fed and trained with the pre-processed train set by using the `model.fit()` method. It is notable that the loss function used for the training of this model is Mean Squared Error (MSE), as opposed to ordinary classification tasks. This is because the ground truth label in the dataset is a scalar value ranging from 0 to 1. The hyperparameters used in the training of the model are as follows.

Epochs	Batch size	Validation split	Optimizer
8	512	0.2	Adam

Table 6-2: Training hyperparameters for the LSTM model.

The evaluation result of the model will be discussed in the next chapter.

6.3.4 Training of BERT Model

Unlike the LSTM model, BERT has its own embedding layer, therefore there is no need to perform pre-processing for the dataset. This makes the implementation of the BERT model fairly simple.

```
def build_bert_classifier_model(bert_model_name):
    tfhub_handle_encoder = map_name_to_handle[bert_model_name]
    tfhub_handle_preprocess = map_model_to_preprocess[bert_model_name]

    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, name='BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']

    net = tf.keras.layers.Dense(384, activation='relu')(net)
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(1, activation='sigmoid', name='classifier')(net)
    return tf.keras.Model(text_input, net)

model = build_bert_classifier_model('bert_en_uncased_L-12_H-768_A-12')
model.summary()
```

Figure 6-9: Source code for constructing BERT profanity detection model.

By leveraging TensorFlow Hub, the BERT model can be rebuilt in just a few lines of code. TensorFlow Hub is an open repository and library for reusable machine learning. There are tons of pre-trained models hosted on this repository. To access it from the notebook document, 'hub.KerasLayer(string)' is used. This method will load the model from the repository based on the handle specified. In this case, the handle would be 'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3' for the Bert encoder and 'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3' for the preprocessing layer.

```
batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE

train_ds = (
    tf.data.Dataset
        .from_tensor_slices((x_train, y_train))
        .shuffle(2048)
        .batch(batch_size)
        .prefetch(AUTOTUNE)
)
```

Figure 6-10: Converting dataset in Pandas DataFrame to TensorFlow Dataset.

Due to the fact that the data after the tokenizing or the embedding process of BERT consumes quite a lot of memory, therefore the dataset could not be tokenized altogether and stored in memory first prior to the training process. This is because this can crash the notebook as the out of memory error could occur. Instead, the data can be loaded and processed dynamically parallel with the training process. This can be achieved by using the Dataset API provided by TensorFlow data module as shown in Figure 6-10. Basically, the code is constructing the TensorFlow Dataset by using ‘Dataset.from_tensor_slices()’, then the dataset is configured to randomly sample entries from a buffer with size of 2048 using ‘shuffle()’.

```

model = build_bert_classifier_model('bert_en_uncased_L-12_H-768_A-12')
model.summary()

Model: "model_1"
-----
Layer (type)                Output Shape                Param #   Connected to
-----
text (InputLayer)           [(None,)]                   0         []
preprocessing (KerasLayer)   {'input_word_ids': (None, 128),
                          'input_type_ids': (None, 128),
                          'input_mask': (None, 128)}
BERT_encoder (KerasLayer)    {'pooled_output': (None, 768),
                          'default': (None, 768),
                          'encoder_outputs': [(None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768),
                                                (None, 128, 768)],
                          'sequence_output': (None, 128, 768)}
dense_1 (Dense)              (None, 384)                 295296    ['BERT_encoder[0][13]']
dropout_1 (Dropout)          (None, 384)                 0         ['dense_1[0][0]']
classifier (Dense)           (None, 1)                   385       ['dropout_1[0][0]']
-----
Total params: 178,149,122
Trainable params: 295,681
Non-trainable params: 177,853,441

```

Figure 6-11: Model summary by TensorFlow for the proposed BERT profanity detection model.

At this point, all setup is completed and the BERT model is ready to be trained. Based on Figure 6-11, there are a total of 295,681 trainable parameters in the model. The weights for the embeddings layer of the BERT and BERT itself are prohibited from updating. The model is trained by using the ‘model.fit()’ method with hyperparameters set as the following table. These hyperparameters are suggested by the original author of BERT.

Epochs	Batch size	Learning rate
3	16	Adam (3×10^{-5})

Table 6-3: Training parameters of BERT profanity detection model.

The trained model will be tested and evaluated on the test set, and its performance metrics will be discussed in the next chapter, Chapter 7.

6.4 Development of DeepProfane Package

At this stage, the research part of the model is completed. The key thrust of this project now moves into building the DeepProfane package which utilizes the trained profanity detection model.

6.4.1 Model Hosting

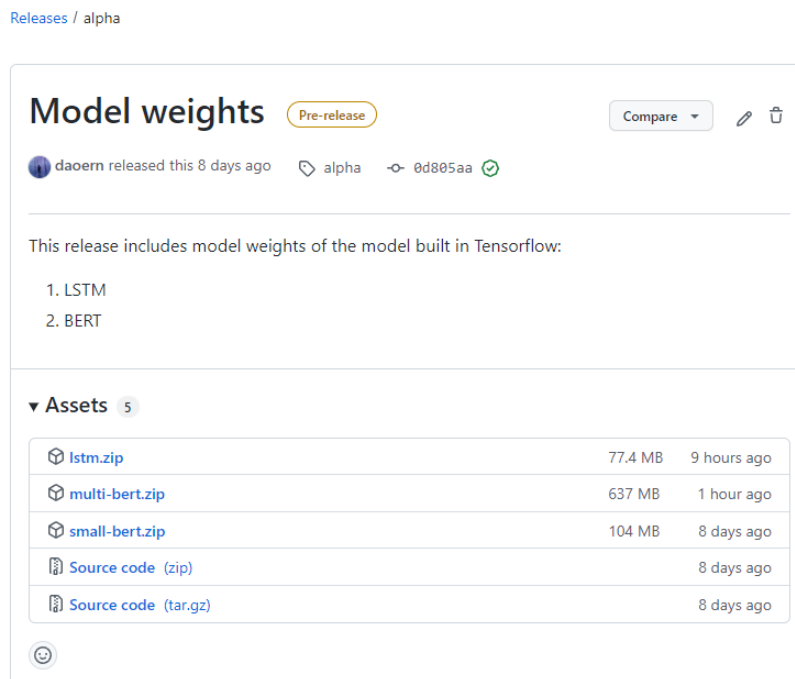


Figure 6-12: Trained model hosted using GitHub Releases.

First, the trained model needs to be hosted at a public server so that it can be accessed by anyone on the Internet. GitHub Releases can be used to host release packages for the users to download. Since this is a free and reliable service, therefore the trained model weights are stored here. The model weights can be accessed using the following link:

- <https://github.com/daoern/deep-profane/releases/tag/alpha>

6.4.2 Packaging as Python Package (deep-profane)

The first module in the DeepProfane package is deep-profane, which is designed to be a Python library published at PyPI. The source code of deep-profane is pushed to a GitHub repository and publicly available at this link:

- <https://github.com/daoern/deep-profane>

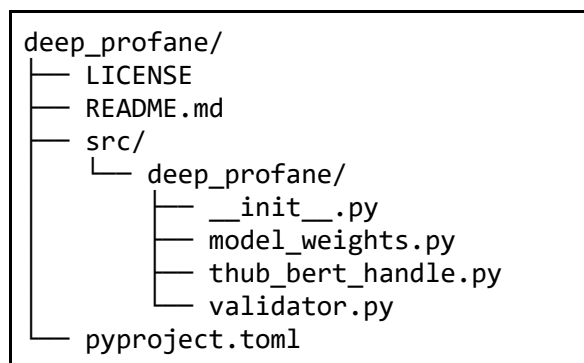


Figure 6-13: The file structure of the deep-profane, Python package.

The file structure for deep-profane is shown in Figure 6-12. First of all, all the main Python source code files are located inside the ‘src/deep_profane’ directory. There are 3 files in this case, which are model_weights.py, thub_bert_handle.py, and validator.py. The file ‘validator.py’ contains the ProfanityValidator class, and ‘model_weights.py’ will handle the loading of model weights from GitHub Releases. Apart from these Python files, ‘pyproject.toml’ is also important as this file specifies the build tools, dependencies, and metadata about how this project is going to be built. In this case, the build tool selected is Hatchling. The full content of this ‘pyproject.toml’ can be viewed at the GitHub repository.

```

1  from . import tfhub_bert_handle
2  from . import model_weights
3
4  # deep learning
5  import tensorflow as tf
6  import tensorflow_text as text
7  import tensorflow_hub as hub
8  from keras.layers import Dense
9
10 You, last week | 1 author (You)
11 class ProfanityValidator:
12     def __init__(self):
13         print("Building model...")
14
15         self.model = self.build_classifier_model('small_bert/bert_en_uncased_L-4_H-512_A-8')
16
17         print("Loading weights...")
18         weight = model_weights.fetch_weights('small-bert')
19         self.model.load_weights(weight)
20
21         print("Model initialized!")
22
23     def build_classifier_model(self, bert_model_name):
24         text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
25         preprocessing_layer = hub.KerasLayer(tfhub_bert_handle.preprocess[bert_model_name], name='preprocessing')
26         encoder_inputs = preprocessing_layer(text_input)
27         encoder = hub.KerasLayer(tfhub_bert_handle.encoder[bert_model_name], trainable=False, name='BERT_encoder')
28         outputs = encoder(encoder_inputs)
29         net = outputs['pooled_output']
30
31         net = tf.keras.layers.Lambda(lambda x: tf.stop_gradient(x))(net)
32
33         net = Dense(512, activation='relu')(net)
34         net = tf.keras.layers.Dropout(0.1)(net)
35         net = tf.keras.layers.Dense(1, activation='sigmoid', name='classifier')(net)
36         return tf.keras.Model(text_input, net)
37
38     def get_profane_prob(self, texts):
39         return self.model.predict(tf.constant(texts)).ravel()
40
41     def is_profane(self, texts):
42         return (self.get_profane_prob(texts) > 0.5).astype(bool)

```

Figure 6-14: Source code for ProfanityValidator in Python.

In ProfanityValidator, the 2 main methods which are `get_profane_prob()` and `is_profane()` that are implemented as shown in line 38, and line 41. Notice that in the initialization function of ProfanityValidator, it will call `model_weights.fetch_weights()` to download trained model weights, then these weights will be loaded into the model.

To publish this Python package to PyPI, the package has to be first compiled into a distribution file. This can be done by using the command, `python -m build`. Once the build is completed, 2 files will be created in the `dist/` directory. The extension of one of the files is `.whl` and another one is `.tar.gz`. After that, this package is ready to be published to PyPI, by using the command `twine upload dist/*`. This command will ask for your PyPI account credentials (username and password). Both of these commands, `build` and `twine` can be installed onto a machine by using `python -m pip install build twine`.

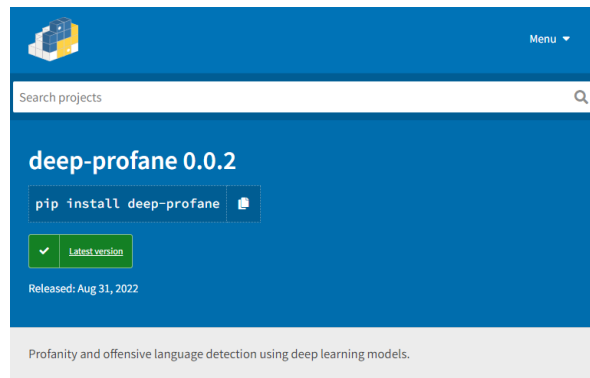


Figure 6-15: The Python package, *deep-profane*, published at PyPI.

After the package is uploaded, *deep-profane* is now available at PyPI.

6.4.3 Serving of RESTful API (*deep-profane-rest*)

Next, the implementation of the RESTful API module of *DeepProfane* package which is called *deep-profane-rest* will be discussed in this section. The source code of *deep-profane-rest* can be found at GitHub repository using the following link:

- <https://github.com/daoern/deep-profane-rest>

```

1  # rest api
2  from flask import Flask
3  from flask_restful import Resource, Api, reqparse
4
5  from deep_profane.validator import ProfanityValidator
6
7  app = Flask(__name__)
8  api = Api(app)
9
10 profane_checker = ProfanityValidator()
11
12 def get_msg():
13     parser = reqparse.RequestParser()
14     parser.add_argument('msg', action='append')
15     args = parser.parse_args()
16     return args["msg"]
17
18 @app.route("/")
19 def home():
20     return "Deep Profane REST API"
21
22 class IsProfane(Resource):
23     def post(self):
24         return {'is_profane': profane_checker.is_profane(get_msg()).tolist()}
25
26 class ProfaneProb(Resource):
27     def post(self):
28         return {'profane_prob': profane_checker.get_profane_prob(get_msg()).tolist()}
29
30 api.add_resource(IsProfane, '/is_profane')
31 api.add_resource(ProfaneProb, '/profane_prob')
32
33 if __name__ == '__main__':
34     app.run(debug=False, host='0.0.0.0', port=80)

```

Figure 6-16: Source code, *app.py* for *deep-profane-rest* in Python, by using *flask*, *flask_restful*.

The source code for deep-profane-rest is fairly simple, as it does not need to handle the computation of the probabilities of profanity by itself. The source code contains only one Python file called 'app.py'. This Python file will start the RESTful server and then send back the profanity detection response to the client based on the messages given at the request. As shown in Figure 4-6, 'app.py' will just forward the request by passing the message list to ProfanityChecker as shown in line 23, and line 27. By using flask, together with flask_restful, the 2 profanity detection endpoints as discussed earlier can be implemented as shown in line 29, and line 30. This server can be started just by using the command to run a Python script.

In order to serve this profanity detection RESTful API to the public, a server is required. For the demonstration purposes of this project, a paid server is rented from Vultr. The RESTful API is expected to stay online until October 2022. The API can be accessed by using the following link:

- <http://207.148.125.180:80/>

To set up the server to serve the profanity detection RESTful API. The source code must first be downloaded at the server. Since the source code is hosted at GitHub, the command, 'git clone https://github.com/daoern/deep-profane-rest.git', can be used at the console to clone the code. Once the source code is available, switch directory into the deep-profane-rest folder by using 'cd' command. Then, install all dependencies of deep-profane-rest by using 'pip install -r requirements.txt'. Last but not least, the server can now be started successfully by using 'python3 app.py'.


```

root@vultr:~/webapp/deep-profane-rest# python3 app.py
2022-09-08 07:57:14.445504: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory
2022-09-08 07:57:14.445740: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
Building model...
2022-09-08 07:57:20.827805: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory
2022-09-08 07:57:20.828051: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)
2022-09-08 07:57:20.828198: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] Kernel driver does not appear to be running on this host (vultr): /proc/driver/nvidia/version does not exist
2022-09-08 07:57:20.832067: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-09-08 07:57:40.119185: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 62509056 exceeds 10% of free system memory.
Loading weights...
2022-09-08 07:57:41.498892: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 62509056 exceeds 10% of free system memory.
Model initialized!
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:80
* Running on http://207.148.125.180:80
INFO:werkzeug:Press CTRL+C to quit

```

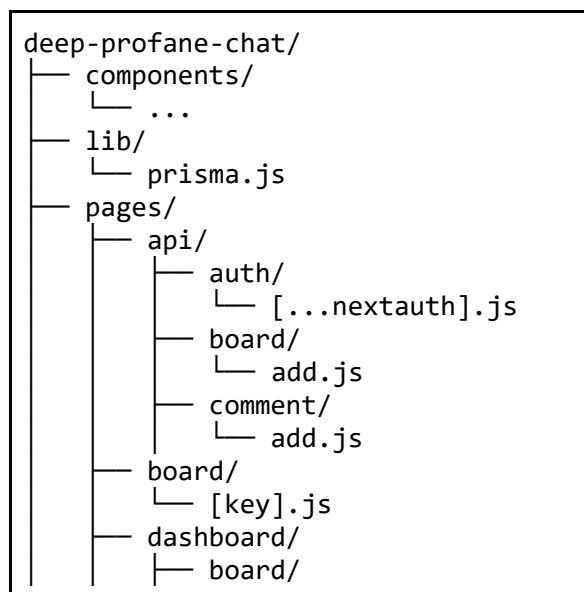
Figure 6-17: Screenshot of the running of the deep-profane-rest on Vultr server.

Notice that in Figure 6-17, the server is now running on `http://207.140.125.180:80` as shown in the second last line of the console output.

6.4.4 Deployment of Web Chatting Application (DeepProfane Chat)

In this section, the development of DeepProfane Chat is documented. The full source code of DeepProfane Chat is publicly available on GitHub, and can be found in the following link:

- <https://github.com/daoern/deep-profane-chat>



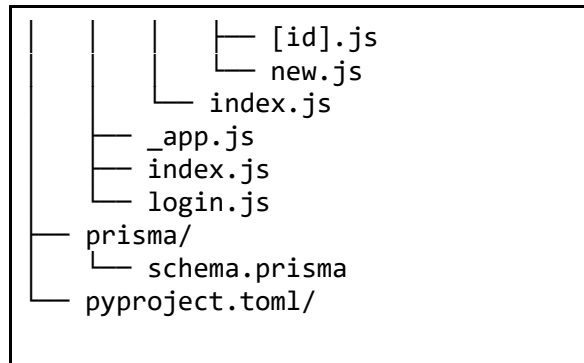


Figure 6-18: The simplified file structure of the deep-profane-chat.

There are many files inside the source code of DeepProfane Chat. Only the key components to get DeepProfane Chat running will be discussed at this section. The key components are displayed in Figure 6-13.

```

13 model Board {
14     id      Int      @id @default(autoincrement())
15     key     String   @unique
16     name    String
17     title   String?
18     createdAt DateTime @default(now()) @map(name: "created_at")
19     owner   User     @relation(fields: [ownerEmail], references: [email])
20     ownerEmail String
21     comments Comment[]
22     domain  String?
23 }
24
25 model Comment {
26     id      Int      @id @default(autoincrement())
27     boardId Int
28     board   Board    @relation(fields: [boardId], references: [id])
29     userEmail String
30     user    User     @relation(fields: [userEmail], references: [email])
31     content String?
32     createdAt DateTime @default(now()) @map(name: "created_at")
33     profane Boolean   @default(false)
34     deleted Boolean   @default(false)
35 }
36
37 model User {
38     email    String    @id @unique
39     name     String?
40     createdAt DateTime @default(now()) @map(name: "created_at")
41     updatedAt DateTime @updatedAt @map(name: "updated_at")
42     comments Comment[]
43     @map(name: "users")
44     boards  Board[]
45 }
  
```

Figure 6-19: The schema definition inside 'schema.prisma' file.

The CRUD operation to the database is done via an ORM library called Prisma. Therefore, a Prisma configuration file named 'schema.prisma' is created inside the 'prisma/' directory. As shown in Figure 6-19, the 3 tables, namely User, Board, and Comment are defined together with their relational constraints.

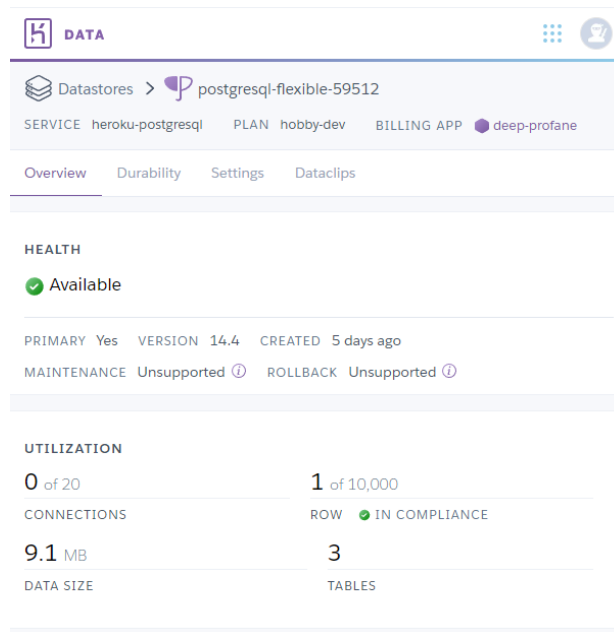


Figure 6-20: Heroku Postgres for deep-profane-chat.

A Heroku Postgres is created and used to host the database of DeepProfane Chat. The credentials of the Heroku database is available in the settings tab. The URI to the database is copied into the ‘.env’ file, which is saved as an environment variable named ‘DATABASE_URL’. Prisma will look for this URI during the starting of this program to connect to the database.

```

1  import NextAuth from "next-auth";
2  import GoogleProvider from "next-auth/providers/google";
3  import prisma from "../../lib/prisma";
4  export default NextAuth({
5    // Configure one or more authentication providers
6    providers: [
7      GoogleProvider({
8        clientId: process.env.GOOGLE_ID,
9        clientSecret: process.env.GOOGLE_SECRET,
10     }),
11     // ...add more providers here
12   ],
13   callbacks: {
14     > async signIn({ user, account, profile, email, credentials }) { ...
30     },
31   },
32   > cookies: { ...
59   },
60 });

```

Figure 6-21: Specifying Google OAuth Provider by using NextAuth.js.

Client ID for Web application [DOWNLOAD JSON](#) [RESET SECRET](#)

Name *
Deep Profane Chat

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorized domains](#).

Authorized JavaScript origins ⓘ
For use with requests from a browser

[+ ADD URI](#)

Authorized redirect URIs ⓘ
For use with requests from a web server

URIs 1 *
http://localhost:3000/api/auth/callback/google

URIs 2 *
https://deep-profane.herokuapp.com/api/auth/callback/google

[+ ADD URI](#)

Note: It may take 5 minutes to a few hours for settings to take effect

[SAVE](#) [CANCEL](#)

Figure 6-22: Setting up of OAuth 2.0 client for DeepProfane Chat at Google Cloud Console.

Implementing the authentication service is straightforward with the help of NextAuth.js library. First, as displayed in Figure 6-21, the GoogleProvider is declared at line 6, with `clientId` and `clientSecret` set as the values from the environment variables. Then, the `clientId` and `clientSecret` can be obtained by registering the app at Google Cloud Console for OAuth 2.0 as shown in Figure 6-22. Notice that under the section ‘Authorized redirect URIs’, 2 URIs are set, one is localhost, and another one is herokuapp.com. This authorized redirect URI is required in order for Google OAuth to allow authentication requests from the listed domain. In this case, localhost is intended for deployment for testing purposes on the local machine, while herokuapp.com is for the final deployment on Heroku.

```

16  const response = await fetch("http://207.148.125.180/is_profane", {
17    method: "POST",
18    headers: { "Content-Type": "application/json" },
19    body: JSON.stringify({ msg: [comment] }),
20  });

```

Figure 6-23: Code snippet to perform profanity detection using deep-profane-rest.

In order to filter profane messages published on the platform automatically, the `deep-profane-rest` module is employed. As illustrated in Figure 6-23, this code is used in `'pages/api/comment/add.js'`. The comment is formatted as a JSON request and then sent to the endpoint `'is_profane'`. Then, the response sent back by `deep-profane-rest` will contain the value of either `'true'` or `'false'` to indicate if the message is profane.

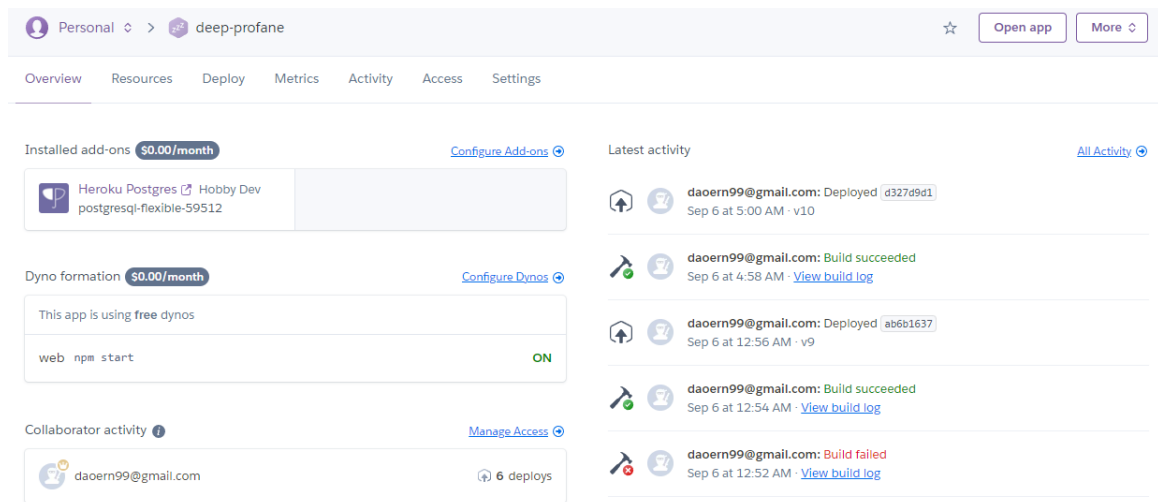


Figure 6-24: Overview of DeepProfane Chat web app on Heroku dashboard.

For the deployment of DeepProfane Chat, Heroku provides deployment of the app with Git. By using Heroku CLI, first run the command `'heroku login'` at the terminal. This will grant you the privileges to access Heroku services. Then, use the command `'heroku create deep-profane-chat'` to start a free-tiered Heroku Dyno project. Next, the last step is to push the code to Heroku by using `'git push heroku main'`. After that, DeepProfane Chat will be up and running as the build and deployment succeeded as shown in Figure 6-24.

6.5 System Operation

The research part of this project does not contain any interactive interface that can be used by end users. Therefore, this section will focus on demonstrating the system operation of the DeepProfane package.

```

%pip install deep-profane==0.0.3

from deep_profane.validator import ProfanityValidator
validator = ProfanityValidator()

Building model...
Loading weights...
100%|██████████| 109M/109M [00:01<00:00, 90.1MiB/s]
Model initialized!

validator.get_profane_prob(["fuck you", "good morning"])

array([0.9358406 , 0.03667727], dtype=float32)

```

Figure 6-25: Usage of deep-profane module.

The users can perform profanity detection easily with the help of the deep-profane module. Since the module is published on PyPI, the command ‘pip install deep-profane’ can be used to install the library onto the machine. Then, simply initialize ProfanityValidator, and use it to perform profanity detection by using get_profane_prob() or is_profane(). As shown in Figure 6-25, deep-profane is very powerful as it enables the users to perform profanity detection with just a few lines of code. The deep-profane module hides the complicated implementation of the model away from the users. This makes it easy to use and accessible by an average developer with no deep learning background.

The screenshot shows a REST client interface for the endpoint `http://207.148.125.180:80/profane_prob`. The request is a POST with a JSON body:

```

{
  "msg": [
    "fuck you",
    "good morning"
  ]
}

```

The response is a JSON object:

```

{
  "profane_prob": [
    0.9358406066894531,
    0.036677274852991104
  ]
}

```

Figure 6-26: The request and response of deep-profane-rest for profane_prob endpoint.

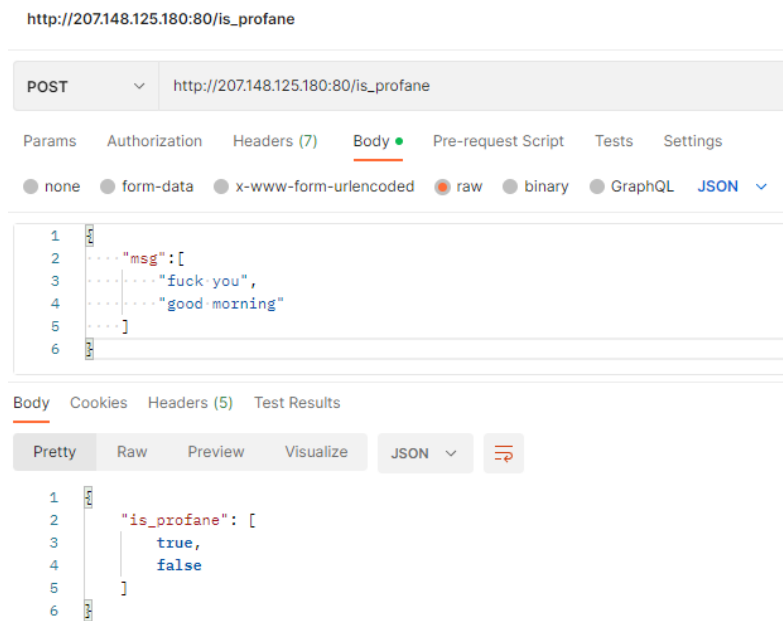


Figure 6-27: The request and response of deep-profane-rest for `is_profane` endpoint.

This deep-profane-rest module, which is hosted publicly, can be accessed at `http://207.148.125.180`. This module provides even greater flexibility compared to deep-profane, as it uses HTTP to communicate and it is not constrained by only the Python programming language. As shown in Figure 6-27, system is tested using Postman by sending two sample messages where one is profane (fuck you), and another one is not (good morning). The results are shown in Figure 6-26 and 6-27. Notice that the API is able to correctly identify the profane message, therefore proving the proposed profanity detection system is feasible.

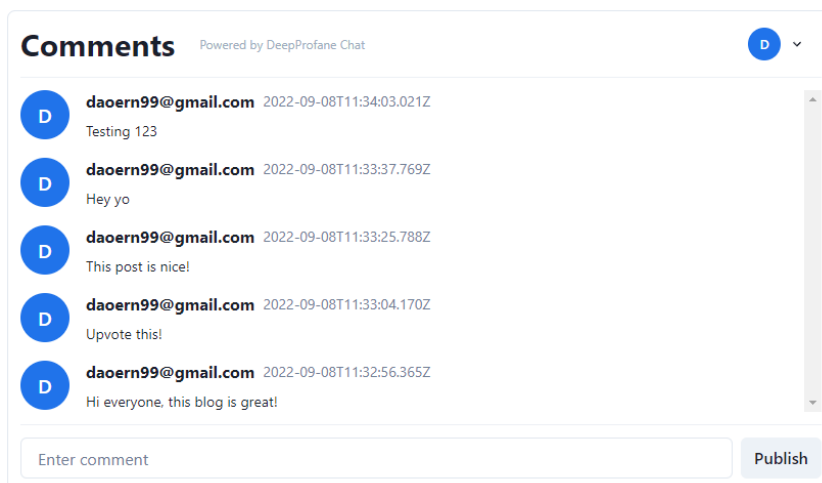


Figure 6-28: The comments board of DeepProfane Chat embedded inside a website.

The last module in the DeepProfane package suite is DeepProfane Chat. The chatting interface as shown in Figure 6-28 is embedded in a blogger website. DeepProfane Chat cannot be used as a standalone chatting system.

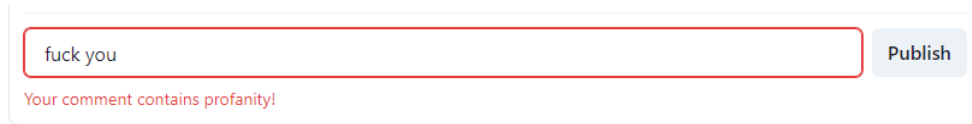


Figure 6-29: Profanity detection in action in DeepProfane Chat.

If the users make a profane comment in the DeepProfane Chat, the comment will be blocked and flagged. A warning will also be issued to the users, preventing the comment from being published. This is demonstrated in Figure 6-29.

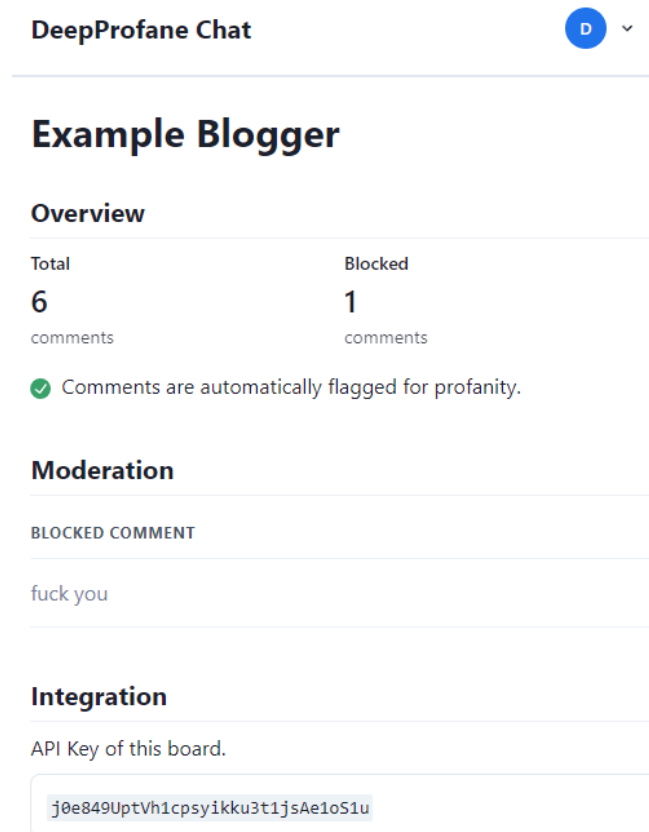


Figure 6-30: Admin dashboard for the comment board shown in Figure 6-28.

The blocked message will not be displayed at the chat box, but it will be shown at the dashboard for the admin to review, as shown in Figure 6-30.

6.6 Implementation Issues and Challenges

The most challenging part of the implementation of the system is to serve the proposed profanity detection model as RESTful API. This is because serving a publicly accessible API requires a server. While most of the cloud solution providers offer a free tier of their hosting services, the computing power and memory provided is limited to run a deep learning model. Therefore, a lot of time has been wasted trying out different platforms to look for stable yet free hosting services. Unfortunately, to the best of the research knowledge, there is no such suitable cloud solution which fulfills the requirement to host and run deep learning models for free. At the end, Vultr, a cloud solution provider is selected for hosting the RESTful API, and it is a paid service.

Another notable challenge in the development of the system is the file size of the trained model. The file size of the trained model ranges up to 600 MB. This size exceeded the maximum package size limit of 60 MB set by PyPI. This means the weights of the trained model could not be packaged and distributed together with the Python package. Instead, the model file should be hosted externally at a file hosting site. Fortunately, GitHub Releases can be used exactly for this task.

The last challenge faced is the lack of computing power during the training of the BERT model. Unlike LSTM, BERT has way more parameters compared to LSTM. Even with the BERT layer set as not trainable and only the last few classifier layers set as trainable, there is still a lot of computation during the forward propagation. Together with the huge dataset used in training, the time for a training epoch of the BERT model is more than 5 hours. This time is measured based on the BERT running on a GPU-enabled notebook on Google Colab. Moreover, since Google Colab is used for training, and training time is long, the training process can easily get terminated by Google Colab halfway due to usage limitations. This makes it more difficult to get the model trained, as it requires splitting the training process into multiple Google Colab session.

CHAPTER 7: SYSTEM EVALUATION

This chapter recorded all testing and evaluation on the system. The evaluation mainly focuses on the performance of the profanity detection deep learning model, but not the DeepProfane package.

7.1 Performance Metrics

Multiple performance metrics are used to evaluate the trained model in this project. Each metric has its own interpretation and all metrics used are discussed in this section.

7.1.1 Accuracy

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{All samples}}$$

Accuracy is the most straightforward metric. It is calculated by dividing the number of correctly predicted samples with the total number of samples. Accuracy is a scalar value which ranges from 0 to 1. The higher, the better the model. However, there is a drawback when using accuracy as the performance metric. That is, high accuracy of the model does not necessarily mean the model performs well. This is because class imbalance might appear in the dataset, for example 90% of the samples is true. Then, any model can randomly guess the class for the samples and still achieve an accuracy of 90%. Therefore, the accuracy metric can only be considered as an overview of the performance of the model.

7.1.2 Precision, Recall, and F1-Score

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

The next performance metric is precision and recall. Precision measures the percentage of true samples out of all samples predicted as true. In other words, high precision of the model means the positive prediction made by the model has a high chance that the prediction is correct. This means there are less false positives in the prediction. Therefore, the higher the precision of a model, the better.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Next, recall is also used as one of the performance metrics. Recall is computed by taking the number of true positives and then divided by the number of true samples in the dataset. Recall measures the ability of the model to capture all positive samples in the dataset. A high recall of the model means there are less false negatives in the prediction. The higher the recall, the better.

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

By definition, F1 score is the harmonic mean of precision and recall. In the most simple terms, the higher the F1 scores, the better. Generally, F1 score can be used to compare the performance between 2 models as it represents both precision and recall together in a single metric.

7.1.3 Precision-Recall Curve

In a classification task, the output of the model is usually a scalar value which ranges from 0 to 1. It can also be interpreted as the probability of the sample being a particular class. Therefore, a threshold value can be set to classify the sample. For example, the threshold is usually set as 0.5, meaning if a sample is predicted with a probability of more than 0.5, then it will be classified as true. The precision and recall of the model will differ across different values of threshold, as the number of samples predicted as positive or negative can vary. These differences of precision and recall at different thresholds can be plotted as a chart called precision-recall curve.

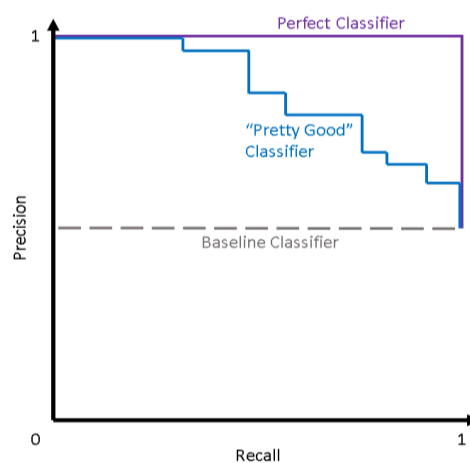


Figure 7-1: Example precision-recall curve [16].

As displayed in Figure 7-1, in theory, a perfect classifier should have the precision-recall curve maxed out at top-right corner, forming a square-like plot. However, this is very difficult to achieve in practice, the usual curve should look more or less like the blue line in the figure. The closer the curve to the top-right corner, the better. Another interpretation of the precision-recall curve is its area under curve (AUC). AUC can be used to generalize the performance of the model across all different thresholds. The AUC of a perfect classifier is 1 since the area is basically a square ($1 \times 1 = 1$). Therefore, the higher the AUC, the better the model.

7.1.4 Receiver Operating Characteristic Curve

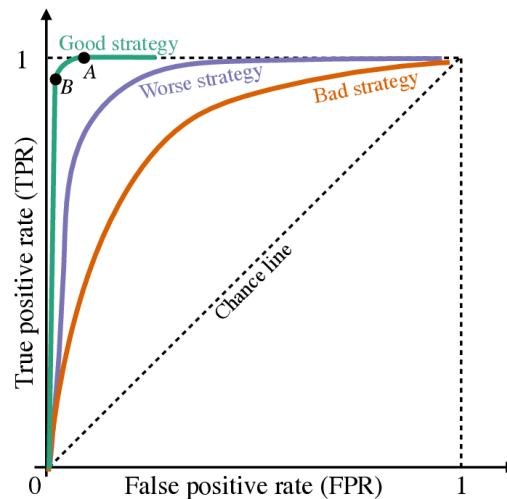


Figure 7-2: Example of Receiver Operating Characteristic.

Similar to the precision-recall curve, Receiver Operating Characteristic (ROC) curve also summarizes the performance of the model at different thresholds. However, ROC shows the trade-off between true positive rate (TPR) and false positive rate (FPR). TPR is also known as sensitivity or recall as mentioned in the previous section, while FPR is a measure of the probability of the model incorrectly classifying the true negative samples as false. In short, the performance of a model is better if the ROC curve is closer to the top left corner.

7.2 Testing Result of the Proposed Model

There are 3 different models produced in this project, which are LSTM, English BERT, and Multilingual BERT. Their performance results are compared and discussed in this section.

Metric	Model		
	LSTM	English BERT	Multilingual BERT
Accuracy	0.9519	0.9266	0.8760
Precision	0.90	0.83	0.56
Recall	0.74	0.56	0.56
F1 score	0.80	0.59	0.56
PR curve (AUC)	0.7733	0.4222	0.1563
ROC curve (AUC)	0.9650	0.8478	0.6944

Table 7-1: Performance of LSTM, English BERT, and Multilingual BERT on the test set.

The test performed on the model is based on the test set of Jigsaw Unintended Bias in Toxicity Classification Challenge. In general, the performance of all the proposed models look average. The metrics recorded by both BERT models show that both of them are way underperformed.

The accuracy for LSTM, English BERT, and Multilingual BERT are 0.9519, 0.9266, and 0.8760, respectively. The accuracy is considered satisfactory, but it could not represent the performance of the model accurately as explained earlier. By looking into the precision, recall, and F1 score, it is noticeable that LSTM performs even better than English BERT and Multilingual BERT. The F1 score for LSTM is 0.8, however it is only 0.59 for English BERT, and 0.56 for Multilingual BERT. This should not be the case since BERT is considered to be a more powerful model. A possible reason for this to happen is due to the limitation of computing power, which has caused the training to be terminated halfway, this could affect the effectiveness of the training of the model.

With these being said, the best possible profanity detection model can be produced in this paper is the LSTM based model. The following figures show the plot of the metrics for the LSTM model.

Although LSTM recorded higher performance results based on these metrics, it is not capable of achieving the understanding of the context of words used in the sentence. Interestingly, BERT models which recorded lower performance metrics are able to understand the context as demonstrated in the next sub-section.

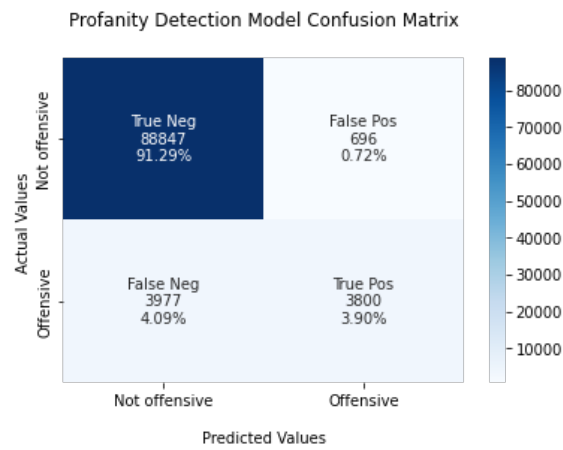


Figure 7-3: Confusion matrix for LSTM model.

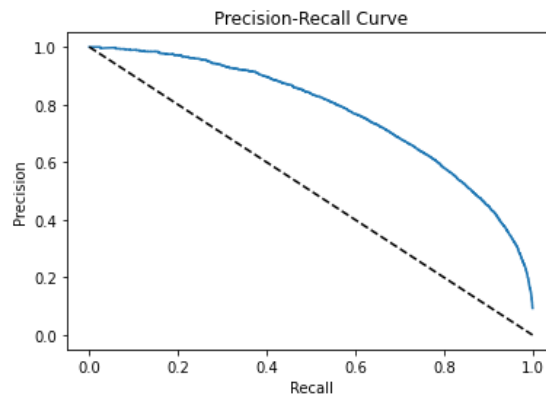


Figure 7-4: Precision-recall curve for LSTM model.

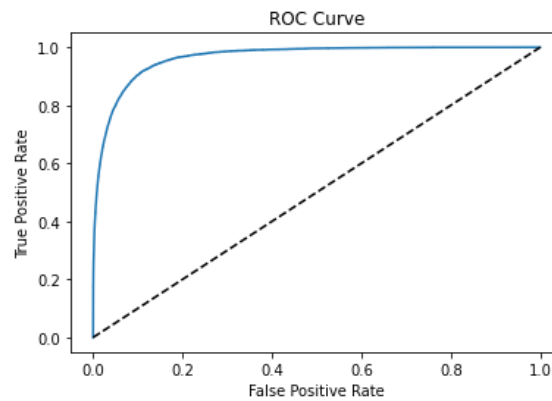


Figure 7-5: ROC curve for LSTM model.

7.3 Objectives Evaluation

```
print(model.predict(tf.constant(['you are suck'])))
tf.Tensor([[0.66403717]], shape=(1, 1), dtype=float32)

print(model.predict(tf.constant(['i suck my thumb'])))
tf.Tensor([[0.16067156]], shape=(1, 1), dtype=float32)
```

Figure 7-6: Output of the English BERT model for the sentences ‘you are suck’ and ‘i suck my thumb’.

Overall, all of the objectives of this project have been achieved. First, the objective is to produce a profanity detection model that is capable of understanding the context of the use in the given sentence. Surprisingly, although the performance metric of the models looks bad, this objective is achieved by using the English BERT model. As shown in Figure 7-6, the probability of profanity for the first sentence ‘you are suck’ is way higher compared to ‘i suck my thumb’, even though the word ‘suck’ appears in both sentences.

```
model.predict(tf.constant(['fuck you', 'пошел на хуй', 'hi everyone']))
1/1 [=====] - 0s 49ms/step
array([[0.7195737 ],
       [0.7697285 ],
       [0.18345378]], dtype=float32)
```

Figure 7-7: Output of the Multilingual BERT model with the first input as English profane text, second as Russian profane text, and third as English non-profane text.

The second objective in this project is to support multiple languages in the profanity detection model. As shown in Figure 7-7, this has also been achieved. The model is able to correctly identify profanity in multiple languages, notice that the first 2 inputs are profane messages in English and Russian. The predicted probability of profanity is high, which is classified as profane. Meanwhile, the last message which is non-profane has low probability.

Next, the third objective is to deploy the model as a RESTful API, so that it is publicly accessible by everyone. This has also been achieved in the module called deep-profane-rest as demonstrated in previous chapters. The sample applies to the last objective, which is to provide a chatting system with automatic profanity detection. This objective is exactly achieved by the module developed called DeepProfane Chat, its working and development is documented in the previous chapter.

7.4 Concluding Remarks

Although there are flaws in the proposed model, and the performance result of the model may not be great, this project is still completed and finished every deliverable as a whole. The tools called DeepProfane provided in this project are especially useful and relevant to the real-world use-case. From the perspective of end users, DeepProfane provided them a tool to fight against profanity easily, by replacing the commenting system on their website with DeepProfane Chat, which is profanity detection enabled. Any website owner or admin can set up this in no time as it does not require any technical knowledge, only a few clicks are needed.

With these being said, this serves as the motivation for the researcher of this project to continue to improve the proposed model and make DeepProfane can be put into greater use in the future.

CHAPTER 8: CONCLUSION

8.1 Conclusion

Abusive use of language has been a problem for Internet users for a long time. Websites have been using some form of content moderation as one of the ways to tackle this issue. Even so, with 4 billion Internet users worldwide, it is difficult to filter out all the offensive content manually by humans as the amount of content generated is too large. Therefore, computers are used to help identify toxic content.

Although the computer can help process huge amounts of data with a shorter amount of time, the computer still could sometimes interpret the data wrongly. This could cause some explicit contents to slip under the radar, or good contents being blocked. For the case of profanity detection in text messages, most of the existing system uses the dictionary-based approach to perform detection. There are a few drawbacks with this approach that have to be solved. First, the Internet users intentionally alternate the spellings of the profane words to bypass the detection. Second, the context for some of the words is important as the word will have very different meanings in different cases. For example, “I suck my thumb.” and “You are suck.”. Third, the dictionary-based profanity checkers are usually limited to only one language, like English, but the messages on the Internet can contain various languages.

With all of that, the ultimate goal of this project is to develop a profanity detection system that is complex enough to correctly identify all offensive messages. The solution proposed in this project is to build a deep learning model to perform profanity classification. Then, the model will be deployed as a suite of tools that can be implemented by other websites. The package suite is called DeepProfane, and the tools include a Python package for profanity detection, a RESTful API for profanity detection, and also a web chatting application with profanity detection.

Based on the performance result tested on all the proposed models, LSTM performed the best among all models. This is quite unexpected as BERT is known to be more powerful and able to solve a variety of NLP tasks with great performance. Nevertheless, although the performance metric is average in general, the model is still capable of recognizing the context of words, and also multiple languages in one model.

8.2 Contributions

The contribution of this project is from the development part of the project. The source code of the DeepProfane is open-sourced and freely available on GitHub. The access link is listed in the previous chapter. This will help any developer interested in developing profanity detection tools, reducing the work for them to build the tool from scratch.

Apart from that, the deliverable, DeepProfane, which is built and provided by this project is also very relevant and useful in real-world scenarios. The Python package (`deep-profane`), and also the RESTful API (`deep-profane-rest`) can be used by an average developer with no experience in deep learning. They can simply integrate profanity detection into their system just by importing these libraries into their project.

Another notable contribution is this project provides website owners a way to secure their website from profane messages in the comment section. This can be done by embedding DeepProfane Chat into their website. DeepProfane Chat can replace the conventional comment system where there is no profanity filtering.

In short, the key thrust of this project is working towards creating a safer and better digital space by minimizing the occurrences of profanity. All these tools provided in this project are created with this goal in mind. By providing these ready-to-use tools, it is hoped that it can reduce the effort to implement profanity detection, so that more and more will adopt profanity detection.

8.3 Future Works

The future work of this project will mainly focus on the improvements of the model. As the proposed model in this project is underperformed, a better model could be studied and implemented in the future, for example XML-R.

REFERENCES

- [1] Kwon, K. and Gruzd, A., 2017. Is offensive commenting contagious online? Examining public vs interpersonal swearing in response to Donald Trump's YouTube campaign videos. *Internet Research*, 27(4), pp.991-1010.
- [2] Newton, C., 2019. The secret lives of Facebook moderators in America. [online] *The Verge*. Available at: <<https://www.theverge.com/2019/2/25/18229714/cognizant-facebook-content-moderator-interviews-trauma-working-conditions-arizona>> [Accessed 15 August 2021].
- [3] Vincent, J., 2021. AI won't relieve the misery of Facebook's human moderators. [online] *The Verge*. Available at: <<https://www.theverge.com/2019/2/27/18242724/facebook-moderation-ai-artificial-intelligence-platforms>> [Accessed 15 August 2021].
- [4] Yin, D., Xue, Z., Hong, L., Davison, B.D., Kontostathis, A. and Edwards, L., 2009. Detection of harassment on web 2.0. *Proceedings of the Content Analysis in the WEB*, 2, pp.1-7.
- [5] International Telecommunication Union, 2020. Measuring digital development. Facts and figures 2020. [online] ITU Publications. Available at: <<https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2020.pdf>> [Accessed 16 August 2021].
- [6] Sampathkumar, M. and Shwayder, M., 2020. Cyberbullying increases amid coronavirus pandemic. Here's what parents can do. *digitaltrends*, [online] Available at: <<https://www.digitaltrends.com/news/coronavirus-cyberbullying-distance-learning/>> [Accessed 16 August 2021].
- [7] Ranasinghe, T. and Zampieri, M., 2020. Multilingual offensive language identification with cross-lingual embeddings. arXiv preprint arXiv:2010.05324.
- [8] Yang, H. and Lin, C., 2020. TOCP: A Dataset for Chinese Profanity Processing. *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*, pp.6-12.

REFERENCES

- [9] Mubarak, H., Rashed, A., Darwish, K., Samih, Y. and Abdelali, A., 2021. Arabic Offensive Language on Twitter: Analysis and Experiments.
- [10] Kumar, R., Lahiri, B. and Ojha, A., 2021. Aggressive and Offensive Language Identification in Hindi, Bangla, and English: A Comparative Study. SN Computer Science, 2(1).
- [11] Yi, M., Lim, M., Ko, H. and Shin, J., 2021. Method of Profanity Detection Using Word Embedding and LSTM. Mobile Information Systems, 2021, pp.1-9.
- [12] Davidson, T., Warmsley, D., Macy, M. and Weber, I., 2017. Automated Hate Speech Detection and the Problem of Offensive Language. Proceedings of the Eleventh International AAAI Conference on Web and Social Media (ICWSM 2017).
- [13] J. J. Ryan T., "LSTMs explained: A complete, technically accurate, conceptual guide with keras," Medium, 10-Sep-2021. [online]. Available at: <<https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2>> [Accessed 20 Mar 2022].
- [14] Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
- [15] Paul, M., "Distributing Large Files with PyPI Packages" Dampfkraft, 25-May-2020. [online]. Available at: < <https://www.dampfkraft.com/code/distributing-large-files-with-pypi.html>> [Accessed 23 Aug 2022].
- [16] Doug, S., "Precision-Recall Curves." Medium, 20-Sep-2020. [online]. Available at: <<https://medium.com/@douglaspssteen/precision-recall-curves-d32e5b290248>> [Accessed 24 Aug 2022].

APPENDIX

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T2, Y3	Study week no.: 4
Student Name & ID: Lim Dao Ern	
Supervisor: Dr. Jasmina Khaw Yen Min	
Project Title: Multilingual Profanity Detection API using Deep Learning	

1. WORK DONE

Studied on the Encoder, Transformer, and BERT model. Studied how to apply BERT for classification task.

2. WORK TO BE DONE

Propose a profanity detection model based on BERT.

3. PROBLEMS ENCOUNTERED

No problem encountered at this stage.

4. SELF EVALUATION OF THE PROGRESS

So far everything is still on schedule.



 Supervisor's signature



 Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	Study week no.: 6
Student Name & ID: Lim Dao Ern	
Supervisor: Dr. Jasmina Khaw Yen Min	
Project Title: Multilingual Profanity Detection API using Deep Learning	

1. WORK DONE

Proposed 2 BERT models. One supports English, and another one supports multiple languages. Began training the BERT model.

2. WORK TO BE DONE

Deploy and package the model to public. Start the development part of this project.

3. PROBLEMS ENCOUNTERED


Limitation of computing power provided by Google Colab. Training of the BERT model gets terminated halfway.

4. SELF EVALUATION OF THE PROGRESS

So far everything is still on schedule.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	Study week no.: 8
Student Name & ID: Lim Dao Ern	
Supervisor: Dr. Jasmina Khaw Yen Min	
Project Title: Multilingual Profanity Detection API using Deep Learning	

1. WORK DONE

Started development of the project. The Python library, deep-profane, is built and published to PyPI.

2. WORK TO BE DONE

Continue the development to build the remaining module, including RESTful API, and the web app.

3. PROBLEMS ENCOUNTERED

Huge file size

4. SELF EVALUATION OF THE PROGRESS

So far everything is still on schedule.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	Study week no.: 10
Student Name & ID: Lim Dao Ern	
Supervisor: Dr. Jasmina Khaw Yen Min	
Project Title: Multilingual Profanity Detection API using Deep Learning	

1. WORK DONE

RESTful API is implemented and hosted on a server which is publicly available to everyone on the Internet.

2. WORK TO BE DONE

Continue the development to finish the last module, a web chatting application with profanity detection.

3. PROBLEMS ENCOUNTERED


No free server capable of hosting a deep learning model is found. So, the RESTful is deployed to a paid server.

4. SELF EVALUATION OF THE PROGRESS

So far everything is still on schedule.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	Study week no.: 12
Student Name & ID: Lim Dao Ern	
Supervisor: Dr. Jasmina Khaw Yen Min	
Project Title: Multilingual Profanity Detection API using Deep Learning	

1. WORK DONE

Completed the web chatting application, DeepProfane Chat. It is published to Heroku and its up and running.

2. WORK TO BE DONE

Finalize the source code, open source the source code, and compile the report.

3. PROBLEMS ENCOUNTERED


There is no problem encountered this time.

4. SELF EVALUATION OF THE PROGRESS

So far everything is still on schedule. All deliverables are completed.




Supervisor's signature



Student's signature

POSTER





Multilingual Profanity Detection API using Deep Learning

Create a safe and moderated digital space for everyone on the Internet!

Introduction

Profanity is an offensive use of language and it is often associated with abusive intentions to cause psychological harm. To counter the increasing prevalence of profanity in digital spaces, content moderation is one of the solutions to this problem. It is a practice of screening user-generated content, either done manually by humans or automatically by computers.







Motivation

A study shows that there has been a 70% increase in hate speech among teens since the COVID-19 lockdown began. On top of that, the EU regulatory framework, has also require companies to monitor user-generated and take down any hate speech content. Thus, the motivation of this project provide a reliable tool for them to automate the moderation.

Problems

Most of the existing profanity detection systems use the dictionary-based approach, and they have many serious drawbacks. First, users can intentionally alternate the spellings to bypass the detection. Second, they are unable to understand the context of words, making false detection. Third, most of them support only 1 language, but the content could be multilingual.



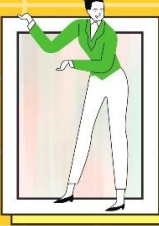


Proposed Solution

Build a profanity detection system using deep learning approach. Currently, the model is proposed to use LSTM with FastText word embeddings. LSTM will act as the feature extractor. The model will be trained, tested, and deployed as REST API, so that it can be accessed by third-party websites, or apps for them to build a moderated space at ease.

Results

At the current stage of development, the proposed solution proved to be feasible. The model achieved a great performance with a testing accuracy of 96.14%. The model will be further improved in Project 2.




PROJECT DEVELOPER


- Lim Dao Ern

PROJECT SUPERVISOR

- Dr Jasmina Khaw Yen Min



PLAGARISM CHECK RESULT



Originality Report

Processed on: 09-Sep-2022 12:14 +08
 ID: 1895648153
 Word Count: 14911
 Submitted: 1

Multilingual Profanity Detection API using De...

By Lim Dao Ern

Similarity by Source	
Internet Sources:	5%
Publications:	4%
Student Papers:	1%

Document Viewer

include_quoted include_bibliography excluding_matches < 8 words mode: show highest matches together Change mode

CHAPTER 1: INTRODUCTION 1.1 Background Information Profanity is an offensive use of language that is deemed impolite and rude. Sometimes, it is also known as cursing, or swearing. Profanity can be viewed as an extreme way to express disrespectful feelings towards something by using abusive language. In most cases, bad words or expletives are used to make a message profane. For each language, there are usually sets of words which are widely recognized by the speaker as expletive. For example, in English, some of the common expletives include "f*ck", "b*tch", "c*nt", etc. By making use of these words, one can construct messages to insult and bully in a verbal manner. Although it is possible for profanity to be used as a signal of informality or close relationship between the speaker and listener, social interactions on digital public platforms usually involve netizens who have distant or no relationship with each other [1]. Therefore, profanity on the Internet is often associated with abusive intention to offend and cause emotional or psychological harm. To counter the increasing prevalence of profanity in digital spaces, content moderation is one of the solutions to this problem. Content moderation is the practice of screening user-generated content which could be presented in a wide range of forms like images, videos, posts, messages, and more. In this paper, monitoring of text messages for profanity is specifically discussed. Profanity detection is a subset of content moderation which allows the system to classify if a message is offensive or not, so that appropriate action could be taken like hiding or censoring the message. Profanity detection not only can be applied to social networks, but also other platforms like community forums (Yin et al., 2009), in-game chatting systems, and sites with comment streams. Figure 1-1 shows a profanity filter blocking a profane message from being sent to public chat on a messaging software, Discord. Figure 1-1: Profanity filter in action on Discord. In general, content moderation can be categorized into manual moderation and automated moderation, or both. Most of the

social networks adopt a combination of automated filtering and human moderators to keep unwanted content off their platforms 13

[3]. For human moderation, it employs humans to manually monitor and screen the content generated by the users. Human moderation may filter the content with high reliability and accuracy, but it has its own limitations. First, humans could fail to moderate in real-time like messages going through the online chatting systems. Secondly, it is not feasible to utilize human moderation when there is a sheer amount of content generated like huge social media sites. This is due to the fact that human moderators often work in highly stressful

- 1 1% match (Internet from 02-Dec-2020)
<https://www.aclweb.org/anthology/2020.emnlp-main.470.pdf>
- 2 1% match (Internet from 31-Oct-2021)
<https://www.hindawi.com/journals/misy/2021>
- 3 < 1% match (Internet from 29-Oct-2021)
<https://www.mdpi.com/2078-2489/12/8/306/htm>
- 4 < 1% match (Internet from 18-Jun-2022)
<https://www.mdpi.com/2079-9292/11/12/1852/html>
- 5 < 1% match (publications)
["Innovations in Smart Cities Applications Volume 4", Springer Science and Business Media LLC, 2021](#)
- 6 < 1% match (Internet from 05-Dec-2021)
<https://dokumen.pub/data-science-crash-course-for-beginners-fundamentals-and-practices-with-python-9781734790146.html>
- 7 < 1% match (publications)
[MoungHo Yi, MyungJin Lim, Hoon Ko, JuHyun Shin. "Method of Profanity Detection Using Word Embedding and LSTM", Mobile Information Systems, 2021](#)
- 8 < 1% match (publications)
[Deepak Suresh Asudani, Naresh Kumar Nagwani, Pradeep Singh. "Exploring the effectiveness of word embedding based deep learning model for improving email classification", Data Technologies and Applications, 2022](#)
- 9 < 1% match (publications)
[Lee Jia Thun, Phoey Lee Teh, Chi-Bin Cheng. "CyberAid: Are your children safe from cyberbullying?", Journal of King Saud](#)

PLAGARISM CHECK RESULT

Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date:	Page No.: 1 of 1



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Lim Dao Ern
ID Number(s)	19ACB06133
Programme / Course	Computer Science
Title of Final Year Project	Multilingual Profanity Detection API using Deep Learning

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)
Overall similarity index: <u>7%</u> % Similarity by source Internet Sources: <u>5</u> % Publications: <u>4</u> % Student Papers: <u>1</u> %	
Number of individual sources listed of more than 3% similarity: <u>0</u>	
Parameters of originality required, and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: Dr. Jasmina Khaw Yen Min

Date: 9th September 2022

Signature of Co-Supervisor

Name: _____

Date: _____

FYP 1 CHECKLIST



UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student ID	19ACB06133
Student Name	Lim Dao Ern
Supervisor Name	Dr. Jasmina Khaw Yen Min

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Plastic Cover (for hardcopy)
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 9th September 2022