VILNIUS GEDIMINAS TECHNICAL UNIVERSITY
FACULTY OF FUNDAMENTAL SCIENCES
DEPARTMENT OF INFORMATION SYSTEMS

Nathan Yorio

HATE SPEECH DETECTION METHOD FOR TELEGRAM

Master's degree Thesis

Information and Technologies Security, state code 6211BX014
Master of Informatics Sciences

Vilnius 2024

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

FACULTY OF FUNDAMENTAL SCIENCES

DEPARTMENT OF INFORMATION SYSTEMS

Nathan Yorio

HATE SPEECH DETECTION METHOD FOR TELEGRAM

Master's degree Thesis

Information and Technologies Security, state code 6211BX014

Master of Informatics Sciences

Supervisor Prof Doctor Nikolaj Goranin [5]

(Title, Name, Surname)

Consultant _____ [5]

(Title, Name, Surname)

Consultant _____ [5]

(Title, Name, Surname)

Vilnius 2024

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY
FACULTY OF FUNDAMENTAL SCIENCES
DEPARTMENT OF INFORMATION SYSTEMS

Informatics Engineering study field

Information and Information Technologies Security study programme, state code 6211BX014

Information and Information Technologies Security specialisation

APPROVED BY
Head of Department
Nikolaj Goranin
2024-05-29

OBJECTIVES FOR MASTER THESIS

No. ITSfmu-22–8753

Vilnius

For student Nathan Andrew Yorio

Master Thesis title: Hate Speech Detection Method for Telegram

Deadline for completion of the final work according to the planned study schedule.

THE OBJECTIVES:

Aim: To create a solution for hate speech recognition in Telegram social network.

Objectives:
1. To perform analysis on hate speech recognition methods, datasets designed for this task, implementations in Telegram and other social media.
2. To design a bot for hate speech recognition in Telegram.
3. To train a ML/DL method (or use the pre-trained method) on hate speech recognition and to integrate into the Telegram bot.
4. To perform bot and method metric evaluation.

Planned result: Telegram bot for hate speech recognition, ML/DL method testing metrics.

Academic Supervisor Professor Nikolaj Goranin

Vilnius Gediminas Technical University
Faculty of Fundamental Sciences
Department of Information Systems

ISBN | ISSN
Copies No. .....
Date .....-.....-.....

Master Degree Studies **Information and Information Technologies Security** study programme Master Graduation Thesis

Title	**Hate Speech Detection Method for Telegram**
Author	**Nathan Andrew Yorio**
Academic supervisor	**Nikolaj Goranin**

Thesis language: English

Annotation

Modern social media platforms now operate at volumes far greater than their predecessors. They must bend to immense government and societal pressure to moderate and review content. Among these forms of content requiring heavy moderation by most platforms is hate speech. Most of these platforms perform some level of automated content filtering in order to meet the demands required from these aforementioned pressures. Telegram, as an exception to this rule, opts for a more ambiguous moderation strategy that awaits human or government request before content is manually reviewed. Because the platform suffers from governance issues, this thesis proposes the novel application of two methods. First, the use of machine learning models trained to detect hate speech applied in the context of a Telegram bot. Second, to analyze the performance of hate speech detection natural language models against data on which they were not trained. The results show not only that this practical implementation with a bot is possible, but also that the use of these models on novel data reveals the impact that training variation can have on performance. This enriches the academic understanding of platform native content detection strategies that would otherwise be proprietary.

Structure:
Introduction
Literature Analysis
Design Proposal
Implementation
Performance Results
Conclusions

Contents:
81 text page
34 pic.
17 table.
70 bibliography ref.

Keywords: machine, learning, content, detection, hate, speech, natural, language, processing, Telegram, bot, python, tensorflow, hugging, face, pytorch, sentiment, social, media, artificial, intelligence, performance, dataset, label

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Informacinių sistemų katedra

ISBN | ISSN
Egz. sk. .....
Data .....-.....-.....

Antrosios pakopos studijų **Informacijos ir informacinių technologijų saugos** programos magistro baigiamasis darbas	
Pavadinimas	**„Neapykantos“ kalbos aptikimo metodas Telegram platformai**
Autorius	**Nathan Andrew Yorio**
Vadovas	**Nikolaj Goranin**

Kalba: anglų

Anotacija

Šiuolaikinės socialinės žiniasklaidos platformos dabar veikia daug didesniu kiekiu nei jų pirmtakai. Jie turi pasilenkti su didžiuliu vyriausybės ir visuomenės spaudimu moderuoti ir peržiūrėti turinį. Tarp šių turinio formų, kurias daugumoje platformų reikia griežtai prižiūrėti, yra neapykantos kurstymas.

Dauguma šių platformų atlieka tam tikro lygio automatizuotą turinio filtravimą, kad atitiktų reikalavimus, kurių reikalaujama dėl minėtų spaudimų. Telegram, kaip šios taisyklės išimtį, pasirenka dviprasmiškesnę moderavimo strategiją, kuri laukia žmogaus ar vyriausybės užklausos prieš peržiūrint turinį rankiniu būdu. Kadangi platforma kenčia nuo valdymo problemų, šioje disertacijoje siūlomas naujas dviejų metodų pritaikymas. Pirma, mašininio mokymosi modelių, išmokytų aptikti neapykantą kurstančią kalbą, naudojimas Telegram roboto kontekste. Antra, išanalizuoti neapykantą kurstančių kalbų aptikimo natūralios kalbos modelių našumą, palyginti su duomenimis, kuriais jie nebuvo mokomi. Rezultatai rodo ne tik tai, kad šis praktinis įgyvendinimas naudojant robotą yra įmanomas, bet ir tai, kad šių modelių naudojimas naujiems duomenims atskleidžia, kokį poveikį mokymo variacijos gali turėti našumui. Tai praturtina akademinį supratimą apie platformos savojo turinio aptikimo strategijas, kurios kitu atveju būtų patentuotos.

Struktūra:
Įvadas
Literatūros analizė
Projektinis pasiūlymas
Įgyvendinimas
Veiklos rezultatai
Išvados

Turinys:
81 teksto puslapis
34 pav.
17 lentelė.
70 bibliografijos nuor.

Prasminiai žodžiai: mokymasis, turinys, aptikimas, neapykanta, kalba, natūralus, kalba, apdorojimas, telegrama, robotas, python, tensorflow, apkabinimas, veidas, pytorch, nuotaikos, socialinė žiniasklaida, dirbtinis, intelektas, našumas, duomenų rinkinys, etiketė

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

Nathan Yorio, 20221403

(Student‘s given name, family name, certificate number)
Faculty of Fundamental Sciences

(Faculty)
Information and Information Technologies Security, ITSfmu-22

(Study programme, academic group no.)

DECLARATION OF AUTHORSHIP IN THE FINAL DEGREE PAPER

May 29. 2024
(Date)

I declare that my Course Project entitled “Hate Speech Detection Method for Telegram” is entirely my own work. I have clearly signalled the presence of quoted or paraphrased material and referenced all sources.

I have acknowledged appropriately any assistance I have received by the following professionals/advisers: Prof Doctor Nikolaj Goranin

The academic supervisor of my Course Project is Prof Doctor Nikolaj Goranin

No contribution of any other person was obtained, nor did I buy my Course Project.

NATHAN ANDREW YORIO

(Signature) (Given name, family name)

Table of Contents

Table of Figures.... 6
Index of Tables.... 7
Abbreviations.... 8
INTRODUCTION.... 9
Investigation object.... 10
Aim and tasks.... 10
Novelty of the topic.... 10
Relevance of the topic.... 10
Research Methodology.... 11
Scientific Value of the Thesis.... 11
1. LITERATURE ANALYSIS.... 12
1.1 Prohibited Content Issues in Social Networks.... 12
1.1.1 Platform Governance and Automation Issues on Large Social Networks.... 12
1.1.2 Social Media Platform Prohibited Content Definitions and Hate Speech.... 13
1.2 Strategies for Detecting Prohibited Content on Social Networks.... 14
1.2.1 Issues Associated With Automating Large Scale Prohibited Content Analysis.... 14
1.2.2 Hash Database Content Matching.... 14
1.2.3 Advancements in Neural Network Training.... 15
1.2.4 Image Recognition and its Associated Complexity.... 16
1.2.5 Viability of Using Natural Language Processing with Bots.... 16
1.3 Bots on Telegram and Other Platforms.... 17
1.3.1 Existing Telegram Bots and Development Tools.... 19
1.4 NLP with Machine Learning Models for Content Detection.... 21
1.4.1 Machine Learning and Sentiment Analysis.... 21
1.4.2 Social Media Datasets Used for Sentiment Analysis.... 22
1.4.3 Tools for AI Development.... 23
1.5 Design Considerations for a Bot That Can Perform Sentiment Analysis.... 26
1.5.1 Use of Pretrained ML model(s).... 26
1.5.2 Use of Unlabeled Datasets.... 27
1.5.3 Use of Labeled Datasets.... 27
1.5.4 Selection Process for Labeled Datasets to Use for Tests.... 28
1.5.5 Use of Multi-Model Testing for Bias Reduction.... 28
1.5.6 Calculations to Determine the Performance Accuracy of Model Sentiment.... 30
1.5.7 Code Hosting for the Bot.... 31
1.5.8 Bot Security Implications.... 32
1.5.9 Communicating With the Bot Through an Interface.... 32
1.6 Conclusions of Chapter 1 – Literature Analysis.... 34
2. DESIGN PROPOSAL.... 35
2.1 High Level Workflow for Sentiment Analysis.... 35
2.2 High Level Workflow for Telegram Bot.... 36
2.3 Combined High Level Workflow for Sentiment Analysis and Telegram Bot.... 37
2.4 Sentiment Analysis Operational Capability Breakdown.... 37
2.5 Sentiment Analysis Requirements Breakdown.... 39
2.6 Sentiment Analysis Architecture Breakdown.... 40
2.6.1 Before Sentiment Analysis.... 40

2.6.2 Sentiment Analysis with an Initial Candidate Test Model....41
2.6.3 Sentiment Analysis with Additional Models....42
2.6.4 Using F-score as a Measure for Performance....42
2.7 Telegram Bot Operational Capability Breakdown....43
2.8 Telegram Bot Requirements Breakdown....44
2.9 Telegram Bot Architecture Breakdown....45
2.9.1 Bot Registration with API....45
2.9.2 Bot Testing Before Adding Sentiment Analysis....46
2.9.3 Bot Testing With Sentiment Analysis Response....47
2.9.4 Additional Unlabeled Dataset Analysis....47
2.10 Conclusions of Chapter 2 – Design Proposal....49
3. IMPLEMENTATION....50
3.1 Dataset Analysis....50
3.1.1 Labeled Test Dataset Selection and Preparation....50
3.1.2 Labeled dataset analysis....52
3.1.3 Unlabeled dataset analysis....53
3.2 Sentiment Analysis....56
3.2.1 Writing Skeleton Sentiment Analysis Code....56
3.2.2 Implementation of One Fully Functional Pre-trained Model....56
3.2.3 Capturing Some Initial Prototype Outputs and Performance Results....58
3.3 Bot Development....59
3.3.1 Bot API registration....59
3.3.2 Testing the Bot Before Adding Sentiment Analysis Features....60
3.4 Implementation of the Sentiment Analysis in Telegram bot....62
3.4.1 Demonstration of Bot Interaction – Single Model Implemented....62
3.4.2 Demonstration of Bot Interaction – Multiple Models With Aggregate Score....65
3.5 Conclusions of Chapter 3 - Implementation....67
4. PERFORMANCE RESULTS....68
4.1 Sentiment Analysis Response Performance Results....68
4.2 Analysis of Results....71
4.3 Conclusions of Chapter 4 – Results....72
4.4 Recommendations for Further Research....73
5. GENERAL CONCLUSIONS....74
REFERENCES....76

Table of Figures

Figure 1: Process of media being hashed and compared against a hash database.....15
Figure 2: High Level Workflow Diagram for Sentiment Analysis.....35
Figure 3: High Level Workflow Diagram for Telegram Bot.....36
Figure 4: Sentiment analysis code and bot workflow diagram.....37
Figure 5: Sentiment analysis operational capabilities diagram.....38
Figure 6: Pre - Sentiment Analysis Activity Diagram.....40
Figure 7: Sentiment Analysis Primary Model Activity Diagram.....41
Figure 8: Functional Decomposition of Accuracy Verification Step.....42
Figure 9: Telegram Bot Operational Capabilities Diagram.....43
Figure 10: Process to Register a Telegram Bot for Use with the Telegram API.....45
Figure 11: Process for Testing the Bot Before Adding Sentiment Analysis.....46
Figure 12: Process for analyzing an unlabeled dataset.....48
Figure 13: Curated Dataset for hate speech detection excerpt.....51
Figure 14: Example of labeled dataset fields.....52
Figure 15: Cooke dataset good labeling example.....52
Figure 16: Normalized labeled message with JSON.....53
Figure 17: Pushshift Telegram dataset NDJSON string.....53
Figure 18: Single isolated message from Pushshift NDJSON string.....53
Figure 19: Pseudocode for message extraction.....54
Figure 20: Example newline isolated messages.....54
Figure 21: Pseudocode for Nth string extraction process.....55
Figure 22: example function for passing data to initialized model.....57
Figure 23: Line by line sentiment analysis using example dataset.....57
Figure 24: comparison script test result output data for Facebook Roberta.....58
Figure 25: Registering a new Telegram bot with the Botfather.....59
Figure 26: Choosing a name for the new Telegram bot.....59
Figure 27: Picture of bot user identity function and sleep status reporting.....60
Figure 28: Picture of simple bot sleep status and start function.....60
Figure 29: Example code showing bot's simple security mechanism.....61
Figure 30: Picture of full bot interaction cycle.....63
Figure 31: Picture of filler text lorem ipsum message length test.....64
Figure 32: Picture of bot processing lorem ipsum filler message maximum length without issue.....64
Figure 33: Image of bot returning bootstrap aggregate sentiment result for not hate speech.....65
Figure 34: Image of bot returning bootstrap aggregate result for probable hate speech.....66

Index of Tables

Table 1 : Companion Table for All Cloud Platforms....18
Table 2: Telegram Bot Examples....19
Table 3: Comparison of Bot and AI Development Tools....21
Table 4: Large Social Media Dataset Examples....22
Table 5: Pre-labeled Hate Speech Dataset Examples....23
Table 6: Comparison of Bot and AI Development Tools....25
Table 7: Table of Sentiment Analysis Entities and Descriptions....38
Table 8: Sentiment analysis capability names and descriptions....39
Table 9: Sentiment Analysis - Functional & Non-Functional Requirements....39
Table 10: Table of Telegram Bot Entities and Descriptions....43
Table 11: Telegram Bot capability names and descriptions....44
Table 12: Telegram Bot - Functional & Non-Functional Requirements....44
Table 13: Sentiment analysis results - multiple models accuracy calculations....68
Table 14: Roberta Hate Speech Dynabench 4 Performance Results F-score....69
Table 15: Twitter 154M Performance Results F-score....70
Table 16: Multilingual Hate Speech Performance Results F-score....70
Table 17: Model-Per-Dataset Processing Times and Additional Notes....70

Abbreviations

AI – Artificial Intelligence;

ANN – Artificial neural network;

API – Application programming interface;

CLI – Command line interface;

GIFCT – Global internet forum to counter terrorism;

IJCSM - Iraqi Journal for Computer Science and Mathematics;

ML – Machine Learning;

NDJSON – Newline-delimited Javascript object notation;

NLP – Natural language processing;

OCR – Optical Character Recognition;

RLHF – Reinforcement learning from human feedback;

SIHD – Shared industry hash database;

ZST – Archive format used for highly compressed text data;

INTRODUCTION

Moderation is an issue that grows proportionally to the amount of content produced. Modern social media platforms now operate at volumes far greater than their predecessors. They must bend to immense government and societal pressure to moderate and review content at rates far beyond reasonable human capacity (Gorwa et al., 2020). Major platforms (Twitter, YouTube, Meta) and consumers of data (Google) all perform some level of automated content filtering (decidedly ethical or not (Llansó, 2020)) in order to meet the demands required from these aforementioned pressures of these various entities (Gorwa et al., 2020). In order to be competitive in this ecosystem it would appear that many social platforms would be interested in performing a similar level of automation. Telegram, as an exception to this rule, opts for a more lax (and somewhat ambiguous) moderation strategy that awaits human or government request before content is manually reviewed (Badiei, 2020; *Telegram FAQ*, n.d.). This is unusual because Telegram channels (holding as many as 200,000 simultaneous members (*Telegram FAQ*, n.d.)) have at times become host to illegal content, extremism, and violent ideology (Baumgartner et al., 2020). Telegram's competitors of similar or larger scale report high use of AI to detect and remove content. Meta, for instance, in their Q2 2022 enforcement report revealed that AI allowed them to take action on 19.3 million instances of content violation (Bickert, 2022). YouTube, as well, claimed in 2019 that 98% of of the videos removed for violent extremism were flagged by machine learning algorithms (*GIFCT*, 2019). In contrast to industry numbers, research has shown that methods like Natural Language Processing (NLP) have success in classifying distinctions in big data from platforms like Telegram (Jarynowski et al., 2021; Shah et al., 2020). To add fuel to this growing understanding, large scope datasets of more than 300 million messages make analyzing the platform's content with AI techniques even more viable (Baumgartner et al., 2020).

Telegram has the handy capability of creating bots for achieving a variety of tasks on the platform from payment processing to automated news (*Telegram APIs*, n.d.). Many users have independently used this bot platform to create their own automated moderation solutions (but not always with open access to their inner workings in mind) (Larsen, 2017/2022; *Miss Rose*, n.d.). It stands to reason that the efficacy of some of these content identification methods could be extrapolated and automated by a bot as well. This paper aims to explore the viability of applying some of these content recognition techniques, independent or commercial, using Telegram's bot platform.

Investigation object

Automated Telegram / social media content detection methods and implementations of those methods.

Aim and tasks

To create a solution for hate speech recognition in the Telegram social network.

1.) To perform analysis on hate speech recognition methods, datasets designed for this task, implementations in Telegram and other social media.

2.) To design a bot for hate speech recognition in Telegram.

3.) To train a ML/DL method (or use the pre-trained method) on hate speech recognition and to integrate into the Telegram bot.

4.) To perform bot and method metric evaluation.

Novelty of the topic

Current popular (non-academic) Telegram moderation bots are either closed-source (*Miss Rose*, n.d.) so that the methods that they use to perform content filtering are completely unknown, or the methods that they use are more primitive than artificial intelligence (like blacklisting certain words or users (Larsen, 2017/2022) , which is limited to the creativity of the administrator of the bot). Much of existing research in this space appears to primarily focus on the ability to detect certain content based on predefined parameters. (with some non-telegram native tools/libraries like Python (Jarynowski et al., 2021; Korotaeva et al., 2018; Shah et al., 2020)). Existing methods are not mainly focused on purpose-built application to something like a bot. Thus, building upon the foundation of existing methods for content detection, an autonomous AI moderation bot, particularly on Telegram, would be a unique application and field test of these technologies. The other novel activity that is not frequently performed is the testing of machine learning models against data upon which they were not originally trained. Testing the performance of these models along with the application via a bot should provide a unique take on this space, and demonstrate the flexibility (or lack thereof) of models to adapt to new use cases.

Relevance of the topic

The number of consistent user-controlled moderation solutions available for Telegram channels is genuinely quite small. Many implementations dedicated to moderation are personal one-

off projects that reuse the same code as more popular bots (Larsen, 2017/2022). To add to this, the number of daily telegram users is growing rapidly. It is one of the top 5 most downloaded apps in the world and has over 700 million monthly active users (*Telegram Press Info*, n.d.). This large user base has a generally unmet need for better prohibited content detection strategy, due to the increasing prevalence of prohibited content in general on the platform (Shah et al., 2020) Telegram itself relies largely on user self-reporting of content violations (*Telegram FAQ*, n.d.) and it has been recognized that moderation is generally one of Telegram's platform governance weak points (Badiei, 2020). Given these factors, automated (particularly platform native and hands-free) moderation is of increasing relevance.

Research Methodology

For the portion of this research that covers machine learning model performance testing and dataset analysis, quantitative methodologies are used for assessment. For the implementation of the Telegram Bot, a practical implementation of the quantitative methodologies is produced and the implementation is discussed with some qualitative properties in mind.

Scientific Value of the Thesis

Not only is there a need for better content moderation on the Telegram platform (Badiei, 2020), but there don't appear to be many openly reviewable purpose-built bots for such tasks. Bots that do exist appear to focus mainly on being moderation assistant chatbots rather than as stand-in replacements for human moderators of content in channels (*Combot*, n.d.; *Miss Rose*, n.d.; *Telegram Bot Daysandbox Bot*, n.d.). There is, of course, the possibility a bot that uses more advanced techniques like AI to detect prohibited content already exists (whether its source is open or not is another matter). However, the lack of an aggregate source of "best-fit" bots for tasks, reveals a gap in public and scholarly understanding of how they operate behind the scenes. An open academic-format review of the creation of an automated content detection bot should provide a unique perspective for both the platform's governance and viability of applied research on the topic.

1. LITERATURE ANALYSIS

The purpose of the literature analysis is to perform a review of the existing technologies and issues surrounding the field of content detection on social media platforms, with a focus on Telegram. It is also intended shed some light on effective methods both for testing performance and for implementation design.

1.1 Prohibited Content Issues in Social Networks

This section aims to look at some of the more apparent issues that seem to face large scale social networks when it comes to the topic of handling prohibited content, with a focus on hate speech in particular.

1.1.1 Platform Governance and Automation Issues on Large Social Networks

It is clear to the layperson on the modern internet that large social media platforms perform some degree of automated content moderation. In 2020, YouTube reported that 98% of the videos removed from their platform for violent extremism were being flagged through machine learning algorithms (Gorwa et al., 2020). Since it is apparent that platforms use ML (machine learning), a form of AI (artificial intelligence), to moderate their content, the logical next step is to deconstruct their rationale for doing so. It is no secret that social media platforms contend with dubious content on a daily basis. Contrary to the notion of good will, there is a large amount of of government pressure on major technology companies to comply with country-by-country legal regulations (Gorwa et al., 2020). Encrypted chat platforms are an exception to this compliance push and Telegram, in particular, has no transparent process through which it acknowledges its compliance with government requests (Badiei, 2020). The reason that much of the platform falls outside of government pressure is due to the fact that Telegram has a heavy focus on private and encrypted communication (*Telegram FAQ*, n.d.). For platforms that have a public facing component (as Telegram does) many organizations have opted for automation to deal with the sheer scale of such content. Maintaining compliance is often beyond the healthy scope of manual intervention (Gorwa et al., 2020). Perhaps automated or "reactive" content moderation has not yet been applied to Telegram due to philosophical concepts of prior-restraint on free speech (Llansó, 2020), but it is unknowable exactly whether or not this is the case. Telegram *does* moderate the content on publicly viewable materials (Badiei, 2020), but the growing scope of the platform suggests that having an automated strategy (similar to what their competitors use and some research has suggested) would

enhance their ability to align with legal compliance and content moderation pressures. Exploring the adaptation of these automated content detection methods to Telegram is what is of interest here.

1.1.2 Social Media Platform Prohibited Content Definitions and Hate Speech

Different platforms have distinct definitions of what they consider to be content against their terms of service. For the purpose of analysis, what exactly counts as prohibited content is less significant than the ability to automatically detect content based on predefined criteria. Other works seem to focus primarily on detection or categorization for a given scenario like specific indicators of prohibited activity based on language characteristics (Shah et al., 2020). Nonetheless, a good starting point for deciding what to look for can be found by observing what social media platforms already define as prohibited. For instance, in the case of Facebook, prohibited content is heavily defined in their 'Community Standards' (Gorwa et al., 2020) and other platforms have similar guidelines. Telegram has a published "Terms of Service" that includes only the following that users must agree not to do (*Terms of Service*, n.d.):

- Use our service to send spam or scam users.
- Promote violence on publicly viewable Telegram channels, bots, etc.
- Post illegal pornographic content on publicly viewable Telegram channels, bots, etc.

As part of Telegram's general stance on avoiding moderation of private groups and chats (Badiei, 2020), it is important to note that their terms of service mentions that only "publicly viewable" Telegram content is considered within the realm of the company's moderation responsibility. Thus, for conducting prohibited content detection research, it seems like other works have focused on analyzing datasets collected from publicly available Telegram channels in particular (Baumgartner et al., 2020) (Jarynowski et al., 2021), which aligns with the relative ease of collection.

In contrast to the Telegram terms of service, the guidelines for prohibited content on platforms with more robust governance systems in place are much clearer. For instance, Meta (Facebook) community guidelines indicate that they will remove any content that is categorized as "hate speech". This, according to their definition includes "direct attacks against people — rather than concepts or institutions— on the basis of what [they] call protected characteristics (PCs)", after which listing many "protected characteristics" of race, sexual orientation, et cetera (*Hate Speech | Meta Transparency Center*, 2024). So, it stands to reason that if Telegram were interested in platform oversight with the fidelity of its peers, it too would have guidelines such as these which are surprisingly absent. Regardless of whether or not hate speech is inherently illegal is less significant

than the implication that it can be an indicator for other illegal or prohibited actions, such as threats or acts of violence offline (Davidson et al., 2017). Thus, it is possible to extrapolate that detecting hate speech aids in the ability to detect these looser prohibited content forms. It will be clearer as to why this is significant in ***section 1.4.*** Because some of the promising research and datasets associated with this specific form of prohibited content, it is the kind that has been chosen for detection in this research.

1.2 Strategies for Detecting Prohibited Content on Social Networks

In order to detect prohibited content, it's important to both analyze the potential issues with that and the different strategies that are currently employed for detection.

1.2.1 Issues Associated With Automating Large Scale Prohibited Content Analysis

An important challenge for creating a system that detects prohibited content is deciding, legally, what data to use for analysis. Large organizations that detect prohibited content on their platforms are often comparing text or images against private and government hash databases of illicit or illegal content as incidents occur (Gorwa et al., 2020). Two such examples are the National Center for Missing and Exploited Children's (United States) hash database of child abuse imagery and and the GIFCTs SIHD of terrorist content (each public and private databases respectively) (Gorwa et al., 2020). Large social networks identify, rehash, and take down this content in real time (such that copies of the prohibited content are not physically stored). Research, unfortunately, has a fundamentally different level of access to such content than the networks actively removing it. Thus, due to the potentially sensitive nature of the content being analyzed, care must be taken to observe it through either a proxy or to collect it in a way that is acceptable for research. One example of this, published in IJCSM (Iraqi Journal for Computer Science and Mathematics), is the scripted crawling of the data of publicly available Telegram channels. Statistics were then collected about the nature of the content observed, without necessarily aggregating and storing observed data locally (Packeer & Kannangara, 2022).

1.2.2 Hash Database Content Matching

One of the major strategies that appears to be employed regularly to check content against a desired list (prohibited or otherwise) is the use of hash tables. These are populated with values produced by feeding content into hash functions which produce strings that are unintelligible from

the original data that they came from. They serve the purpose of obfuscating, checking, and organizing media in a way that differentiates it from its source material (Wong, 2021). These databases can be used to check if there is a match against a prohibited dataset of text or images by comparing metadata about the content stored along with the hash (Shah et al., 2020). Some well known companies (Microsoft, Facebook, Google, Twitter), currently work together to maintain a shared industry hash database of prohibited content (SIHD) and they collectively form a group called the GIFCT (Global Internet Forum to Counter Terrorism) (Gorwa et al., 2020). They have the ability (and regulatory pressures) to check effectively all content uploaded to their platforms for violation of their terms (Gorwa et al., 2020). Applying this to prohibited content detection media in any format can be passed through a hash function to produce a hash string. This hash string (and its metadata) can be compared with strings in the database to determine whether or not its origins are prohibited content (see ***Figure 1***).

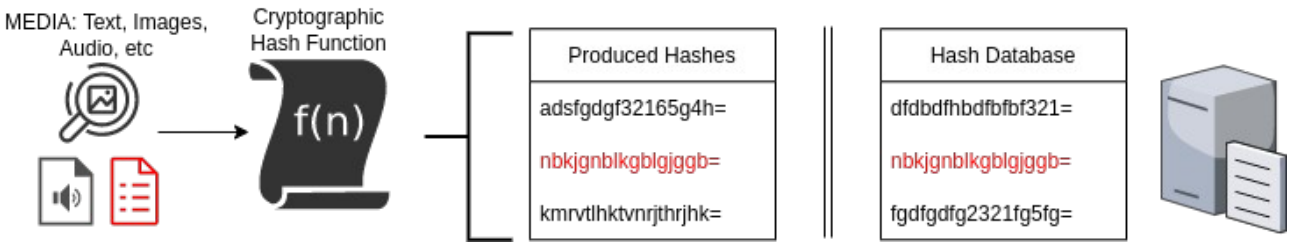

Figure 1: Process of media being hashed and compared against a hash database

The original media, in red, will be identifiable in the end, but appear indistinguishable from its original form

The major issue that hash databases present for independent research is the fact that researchers do not have the resources and instantaneous response requirements of large companies, like those in the GIFCT. The other issue is that these hash databases for prohibited content recognition are largely secretive (Gorwa et al., 2020) and outside of the ability of the research community's use.

1.2.3 Advancements in Neural Network Training

One increasingly common method for using artificial intelligence to distinguish information about a dataset is the use of neural or "deep" networks, which operate on the principal of having multiple "neuronal" layers that work toward a prediction given some initial parameterized input functions (Jordan & Mitchell, 2015). They begin with an input layer (which accepts initial data). After which come hidden sub-layers, which are often taking advantage of algorithmic properties that can sieve the data closer to a probabilistic result (Nadkarni et al., 2011). An example of one such

algorithmic property is Google's BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019). Google's model takes advantage of binary (true or false) cross-entropy loss which, if paraphrased, means that at each layer of the neural network decidedly incorrect values are lost or replaced during each layer reconstruction (getting closer to an accurate prediction result while being resilient against corruption of the original input) (Vincent et al., 2008). These properties are enabled via what are called "activation functions" in the context of neural networks (Jarynowski et al., 2021). These activation functions serve as a gradient of values for the truth probability of the output of a specific neuron in the network (Godoy, 2022). The aggregation of decisions from each layer, acting as a tiered filter, results in a final prediction output layer. The intermediate neuronal layers are "trained" either manually (with the correct prediction data) or algorithmically with weighting and reabsorption or transformation of the inputs (Vincent et al., 2008).

1.2.4 Image Recognition and its Associated Complexity

Image recognition AI (or computer vision, depending on application), like other forms of neural networks, relies on the same multi-neuron-layered decision strategies previously mentioned. Rather than identifying textual characteristics, it relies on the probabilistic determination of objects based on pixel-level annotations of a feature map of the image at each stage of the network cascade (Wang et al., 2021). Due to the quantity of open image data available on the internet contemporary versions of image recognition methods (like VLAD, Vector of Locally Aggregated Descriptors) are trainable on very large datasets like the Flickr10M dataset of 10 million random images (Jégou et al., 2012). The improvement in graphics processing equipment (originally designed for gaming) and the scope of these large datasets has been a major factor for the improvement of computer vision (Jordan & Mitchell, 2015) . Conversely, there are several issues associated with image recognition not present with text based endeavors. One issue is that most image recognition algorithms are designed to recognize shapes in natural images (Wang et al., 2021). Objects are not "natural", blurry or aren't part of a dataset are unlikely to be properly categorized. Image datasets also suffer from blurriness (occlusion), which makes it more difficult to train algorithms to recognize what is in the image (Wang et al., 2021).

1.2.5 Viability of Using Natural Language Processing with Bots

NLP is one specific method for training an artificial neural network (ANN) that has proven success. Google's BERT is one such example of a neural network model optimized for NLP (Devlin et al., 2019). There are even pre-trained Natural Language Processing Models (Like DeepPavlov for

Russian) that can be used to deterministically categorize the content of text data from Telegram channels (Jarynowski et al., 2021). NLP at its core takes advantage of some fundamental properties of the structure of natural human languages. Many language feature characters that only appear in specific sequences (called n-grams) that can infer a lot about the content of a message without even necessarily knowing the context (Nadkarni et al., 2011). Aforementioned solutions apply these fundamental characteristics to ANNs and there are many more use-case specific methods beyond the scope of this document that can be found in Nadkarni et al (2011) . Much NLP development doesn't appear to be done from scratch and there exist frameworks upon which pre-trained classification models (BERT, GPT, Transformers) can be applied to make development easier and faster ("ChatGPT," 2022)(Devlin et al., 2019).

Observation of published articles on the subject indicates that text-based data is a common choice for analysis. An alternative to the Packeer & Kannangura method is the aggregation of publicly available text to train a recognition algorithm. Shah et al ((2020)) scraped the HTML of 102 different telegram channels (legal and "illicit") to feed into their NLP language model to validate that their method could identify content legality. They were even able to use a dataset from Twitter to compare the origin of content as well with "human comprehensible" differences (Shah et al., 2020). Given the success of these aforementioned methods, choosing a sufficiently large natural language dataset should be adequate for testing the efficacy of a moderation bot. It also stands to reason that some of the complexity associated with other detection methods will not be as difficult to handle in the case of natural language processing.

So, the next logical question that follows the assumption that NLP serves as a viable method for content detection is "how will this content detection method be deployed in practice?" Platforms, as previously discussed, have their own internal methods for content detection deployments, but an external party can't leverage internal tooling to detect content. The next closest thing to internal tooling is API exposure, and from this the idea that perhaps a bot (which leverages this platform API as if it were a user) could serve as the agent for content detection using natural language processing.

1.3 Bots on Telegram and Other Platforms

Since it's decided that bots could be the vehicle for performing content detection, it is important to decompose some of the different fundamentals surrounding bots, what they're typically used for, and how they work in the context of Telegram.

Traditional Applications for Chatbots:

The fundamental purpose of a bot (or chatbot, when used in a chat format) is to act in place of a role that would otherwise be occupied by a human. Businesses, in particular, use bots because of their scalability and reduced cost, in opposition to human alternatives (Raj, 2019). Most bots are operating using some form of *intents* and *entities*. Intents are the purpose for using the bot and entities are the keywords and phrases that the bot is looking for in user messages in order to provide some useful function (Kahn & Das, 2018).

Platforms for Bot Development:

Bots, at their core, are small programs that use an API key to authenticate and communicate with the Telegram infrastructure. In this way, their deployment location is fairly flexible. As long as a machine has access to the internet and can asynchronously run the bot code, it could hypothetically host a Telegram bot. Many modern bots are not being deployed on local hardware. Today's bots are often being deployed on cloud (or serverless) platforms. These platforms themselves often provide tools that can assist in the development of bots, including artificial intelligence (Patil et al., 2017)(their useful companion table included below).

Table 1 : Companion Table for All Cloud Platforms

	KORE	CHATFUEL	Microsoft Bot Framework	Microsoft Azure	Heroku	AWS Lambda	IBM Watson
AI BUILT IN	**Yes**	No	**Yes**	**Yes**	No	No	Yes
Programming needed?	No	No	Yes	Yes	Yes	Yes	Yes
Complexity	High	Low	High	High	High	High	Medium
Setup time	10 min	10 min	1 hour	15 min	2 hours	1 day	4 hours
Pre-study	8 hours	4 hours	8 hours	8 hours	8 hours	16 hours	16 hours
IDE	Built in	Built in	Visual Studio	Built in	Eclipse Atom	Eclipse / CLI	Eclipse
Pros	Extensive	Plug 'n play	Most extensive	Integrated Env	Single dev + deployment	Serverless deployment	High quality of interaction
Cons	Steep learning curve	Limited possibilities	Needs setup and deployment	Preview only	Steep learning curve	Steep learning curve	Limited cross-service integrations

Note. Adapted from Comparative study of cloud platforms to develop a chatbot, Patil et al., 2017

Chatbots that are designed to converse with end users frequently use commercial implementation of NLP to achieve similar results to a natural human conversation. For instance, OpenAI's (AI research

business and nonprofit enterprise) most recent tools use "Reinforcement Learning from Human Feedback" (Uc-Cetina et al., 2022) to deliver this functionality to a variety of different enterprise applications, like Algolia semantic search (with GPT-3) and their own AI chatbot, ChatGPT ("ChatGPT," 2022; *OpenAI, Blog, GPT3-Apps*, 2021). Many of these AI platforms feature an open API (including OpenAI) that can be interfaced with in order the leverage existing technology in order to produce new ways of interacting with existing models. Hugging Face, an AI development company, also provides a public API for their tool "Transformers", which can be used to train machine learning (NLP) models (Transformers, n.d.). From the comparison of cloud platforms and the existence of public APIs for NLP models, one could discern that a bot that needs to employ AI can be deployed anywhere as long as it is capable of talking to both Telegram servers and public AI APIs (assuming publicly hosted AI platforms are the method of choice).

1.3.1 Existing Telegram Bots and Development Tools

Most social media platforms provide some form of API that can interact with content in the platform. Many are familiar with the Twitter API (*Twitter API Documentation*, n.d.), which is frequently used to make bots. Telegram also exposes their API for creating bots using a chat registration interface called BotFather, which provides API tokens that can be used for performing actions on Telegram as a proxy for a user (*Telegram Bots*, n.d.).

Table 2: Telegram Bot Examples

Bot	Function	Open/Closed Source Code	Source
Miss Rose	Chat moderation	Closed, previously open	(*Miss Rose*, n.d.) (Larsen, 2017/2022)
DaySandBox	Anti-spam	Closed, previously open	(*Telegram Bot Daysandbox Bot*, n.d.) (lorien, 2017/2022)
Combot	Anti-Spam	Closed	(*Combot*, n.d.) (*Combot Anti-Spam System*, n.d.)
Botanicum	Tree Identification from images	Open	(Korotaeva et al., 2018) (Koro, 2019/2019)
Protectron	"AI" Chat Moderation	Closed, unknown	@antispamchat, only accessible via telegram directly

Anecdotally, many of the Telegram bots observed while creating this review either did not have open source code, or had code that was previously open source and is now closed (presumably for future monetization). Some non-exhaustive examples are included in *Table 2*. As is clear from the table only one of these bots that has a similar focus to this research is from another research article, of which the focus is on image detection of leaves instead of natural language processing. It's quite apparent that there isn't a lot of research surrounding the concept of deploying a bot that leverages NLP for content detection. However, what is clear is that bots are commonly deployed as a means to perform platform moderation tasks, so even in the event that a standalone bot doesn't exist that uses machine learning to detect prohibited content, it could at the very least serve as beneficial for existing bots to be augmented with this technology to enhance their capabilities.

Bots, for the most part, don't really have specific tools that are required to build them. At their most basic, they simply interact with an API. So long as a language exists which supports easily interacting with a given bot API, that alone should be sufficient for building a chatbot.

Table 3: Comparison of Bot and AI Development Tools

Tool Name	Type (Lib, model, etc)	Platform	Used for	Citation
Telethon	Python Telegram Client	Varies, Open	Library for Telegram API Interaction	(Baumgartner et al., 2020; Jarynowski et al., 2021; *Telethon*, 2016/2023)
Python-telegram-bot	Python Telegram Client	Varies, Open	Library for Telegram API Interaction	(*Python-Telegram-Bot*, n.d.)

Note: This is by no means extensive, there are innumerably more tools for development

For Telegram, there are several open libraries for building bots, but it seems the majority of people are using Python, likely because it is simple and interoperable with other libraries. ***Table 3*** includes some examples of Python libraries for Telegram. Telethon was included in a research article that was found during this review, but the python-telegram-bot library appears very robust and easy to work with. Likely a good candidate for this research both due to its simplicity and ease of integration with other tooling, as will be discussed in ***section 1.4***.

1.4 NLP with Machine Learning Models for Content Detection

Since natural language processing appears to be the most viable method for this research, this section serves to break down some of the different aspects of using natural language processing for content detection. There are aspects that cover both models themselves, but data and tools used to test their performance are covered here.

1.4.1 Machine Learning and Sentiment Analysis

When content detection is performed for qualitative properties against a dataset frequently this is referred to as Sentiment Analysis, or the "attempt[s] to automatically determine sentiment contained in text" (Taboada, 2016). Models that are good at performing data classification (EG. "is this prohibited content or not?") are not inherently the same as models that are good at creating generative human-like response to input (EG. GPT). Binary classification of data, such as assigning labels like "hate speech" or "not hate speech" would fall under this sentiment analysis category of machine learning.

1.4.2 Social Media Datasets Used for Sentiment Analysis

There are not a huge number of open Telegram datasets available at the moment. Many of the datasets that were gathered in academic research are not posted online for re-use (Jarynowski et al., 2021; Shah et al., 2020), but the major (rather large and extensive) dataset that is openly available for research use is The Pushshift Telegram Dataset (Baumgartner et al., 2020), which contains 317 million Messages from 2.2 million unique users across 27.8 thousand public channels. The scope of this single dataset is large enough, in fact, that it can and has been used for a variety of other research as well (Bovet & Grindrod, 2022; Peeters & Willaert, 2022). Datasets don't need to be limited just to Telegram. Twitter happens to also be a common place to harvest data for NLP tasks and for content recognition accuracy comparisons (Pak & Paroubek, 2010; Shah et al., 2020). ***Table 4*** contains a list of some social media datasets, their scope, and references.

Table 4: Large Social Media Dataset Examples

Dataset	Platform	Scope	Reference
Pushshift	Telegram	317M Messages	(Baumgartner et al., 2020)
Pak & Paroubek	Twitter	300,000 text posts	(Pak & Paroubek, 2010)
YFCC100M Multimedia Commons	Yahoo, Flickr	100 Million Images	(*Multimedia Commons - Registry of Open Data on AWS*, n.d.)

These large scale datasets tend to have an issue that they suffer from for sentiment analysis. The issue is that they are not labeled in a simple way for sentiment derivation. This complicates some things because the scope of this research is not to develop a process for dataset labeling. Rather it is for detecting sentiment and sets with labels already applied are the best way to test this. After reviewing many different datasets and fine tuned machine learning models, hate speech labeled datasets stood out as having clean formatting for ease of use with binary classifiers (more on specifics in ***Section 3***). It's not so important, necessarily, that the data being used will come from Telegram itself, as all social media share similar characteristics in slang and use of language.

Table 5: Pre-labeled Hate Speech Dataset Examples

Dataset	Platform	Scope	Reference
Curated Hate Speech Dataset	Multiple	451709 messages	(Mody et al., 2023)
Dynamically generated hate speech dataset	Multiple	~40, 000 messages	(Vidgen et al., 2021)
Multi Platform-Based Hate Speech Detection	Multiple	3000	(Cooke et al., 2023)
Automated Hate Speech Detection	Twitter	25000 tweets	(Davidson et al., 2017)
MMHS150K	Twitter	150,000 tweets image and text	(Gomez et al., 2019)
A Benchmark Dataset for Learning to Intervene in Online Hate Speech	Reddit, Gab	22,324 messages	(Qian et al., 2019)

Table 5 contains some examples of pre-labeled hate speech datasets which are quite robust and have supporting academic papers to back them up. Automated Hate Speech Detection, MMHS150K, and A Benchmark Dataset for Learning to Intervene in Online Hate Speech are all particularly good sets to work with. A more detailed explanation of why these sets are ideal can be found in ***Chapter 3,*** dataset analysis.

1.4.3 Tools for AI Development

It is pertinent to compare some of the tools that are used in developing bots and AI in order to determine what would be most effective for creating an automation bot. The programming language of choice for developing a bot is, again, not necessarily important as they can be created in a large variety of languages (Modrzyk, 2019). What is important is that the language supports libraries and tools that can make communication with both AI and bot platform aspects of development smoother. Python, for instance, was used by multiple research groups for AI development tasks, likely due to its “widely-used” (Korotaeva et al., 2018) purpose-built libraries

like PyTorch (an open-source machine learning framework) or OpenCV (computer vision library) (Modrzyk, 2019; *PyTorch*, n.d.; Shah et al., 2020; Wang et al., 2021). ***Table 6*** includes some tools, their purpose, and in which context they have been used in research.

Table 6: Comparison of Bot and AI Development Tools

Tool Name	Type	Platform	Used for	Citation
MMDetection	PyTorch-Based Toolbox	Varies, Open	Computer Vision Research	(MMDetection Contributors, 2018/2018; Wang et al., 2021)
PyTorch	ML Framework	Varies, Open	Machine learning development	(*PyTorch*, n.d.; Raj, 2019; Wang et al., 2021)
Word2Vec (BOW, Skip-Gram)	model/architecture	Varies, Open	Natural Language Processing	(Shah et al., 2020; *Word2vec | TensorFlow Core*, n.d.)
Scikit-learn	Python Library	Varies, Open	Machine Learning development	(Korotaeva et al., 2018; Raj, 2019)
spaCy	Python NLP Library	Varies, Open	NLP Development, Information Extraction	(Raj, 2019; *spaCy*, n.d.)
Tensorflow (and Hub)	Machine Learning Framework	Tensorflow and others	Data acquisition, model training, etc	(Raj, 2019; *TensorFlow*, n.d.)
NLTK (Natural-Language Toolkit)	Python Modules, Datasets, Tutorials	Varies, Open	Natural Language Processing	(*NLTK :: Natural Language Toolkit*, n.d.; Raj, 2019)
Microsoft LUIS	Cloud-based natural language understanding service	Microsoft Azure Bot Service, Commercial	Natural Language Processing, Chatbot Creation	(*Language Understanding (LUIS) | Microsoft Azure*, n.d.; Patil et al., 2017)
HuggingFace Transformers	ML API for PyTorch, Tensorflow, JAX	HuggingFace, commercial, with free tier	Pre-trained ML model classification	(Transformers, n.d.)
GPT-3	API for NLP	OpenAI, commercial, with free tier	Pre-trained Natural Language Processing Model	(*OpenAI, Blog, GPT3-Apps*, 2021)

Note: This is by no means extensive, there are innumerably more tools for development

Alongside open tools for manually trained machine learning models, there are commercial tools as well that all have free-tiers that shift the burden and expense of model training away from the developer. Such tools include GPT-3, HuggingFace, and Microsoft LUIS (Transformers, n.d.; Language Understanding (LUIS) | Microsoft Azure, n.d.; OpenAI, Blog, GPT3-Apps, 2021). The open APIs promise the fastest stand-up time, but the tradeoff of using these platforms is that the details of what datasets with which their models are trained is mainly undisclosed. It is important to note that contemporary machine learning models are not trained from the ground up with code written entirely from scratch. They make use of platforms, frameworks, and libraries of code optimized for machine learning training tasks to process large amounts of data at scale. The two most apparently successful open source platforms for training machine learning models are Tensorflow, created by Google and optimized to work with hardware designed for machine learning (tensor core technology) (*TensorFlow*, n.d.) and Pytorch (*PyTorch*, n.d.). The integrations that tensor-based frameworks have with various platforms, programming languages (like Python and Javascript), and their optimizations make them ideal candidates for working with both AI and Bots.

1.5 Design Considerations for a Bot That Can Perform Sentiment Analysis

Given the conclusions made about ideal tools for content detection in the previous sections, this section serves to expand upon some of the findings when it comes to implementing the different tools discussed.

1.5.1 Use of Pretrained ML model(s)

If desirable accuracy is expected, training a machine learning model using local hardware is far beyond the scope of this research. However, it's important to note the resources associated with such training in order to disseminate its non-inclusion here. Purchasing hardware outright to train a machine learning model can have significant cost even for small models, particularly due to the use of graphical processing units (GPUs) for computation. As a result of the cost of local hardware, most who are training machine learning models have turned to the use of cloud platform services to achieve these goals. However, despite offering elastic scaling for resources, cloud platform services are still a potentially significant cost, on the order of several dollars per hour per instance, increasing to many thousands when applied at scale to large models (Justus et al., 2018). There are many services that offer either compute resource infrastructure to train machine learning models (*Machine Learning – Amazon Web Services*, n.d.) or APIs / interfaces with which to feed data to

fine-tune models. Many open models can be pulled locally into projects and validated for accuracy without dealing with the code overhead or shouldering the cost associated with initial training. Thus, one requirement is the importing of pre-trained base machine learning model(s) which can be built off of or fine-tuned if necessary.

Additionally, in the case of models from repositories, tensor technology is developed by Google and is designed for working with natural language processing models such as BERT. This allows users to easily create new purpose-built models from existing work. Consequently it is also possible to construct sentiment analysis tools from a large pre-made corpus of models designed for specific tasks available freely online.

1.5.2 Use of Unlabeled Datasets

Unlabeled datasets, such as the Pushshift Telegram dataset, have a very large amount of messages. This large quantity of messages makes them ideal for sampling for training data. The drawback of datasets like these are that they are, genuinely, very large. Based on some sample testing while gathering research the Pushshift set is a 50 gigabyte ZST archive when compressed, and its uncompressed size is well over 700 gigabytes. Archive formats like ZST can't directly be broken up into chunks, but pieces of them can be read as data streams in order to extract only a fraction of the message data. In cases like this the entire dataset is not needed for analysis, only one in every arbitrary Nth message can be selected for decompression, to work with a smaller quantity of data. The other issue with unlabeled data is that it is, of course, unlabeled. So it is difficult to use for sentiment analysis because it has to either be manually labeled or labeled with the aid of automated methods, such as machine learning models (Vidgen et al., 2021).

1.5.3 Use of Labeled Datasets

In order to test sentiment analysis accuracy there needs to be a designation of true positive, true negative, false positive, and false negative (***see section 1.5.6***). For reasonable accuracy a large scale dataset needs to have labeling manually applied for classification. For the accuracy of sentiment analysis testing, a pre-labeled dataset is required. Many of these are already used to train models in repositories and can be re-used for the purpose of testing accuracy at this scale as well. This doesn't exclude unlabeled datasets for use in testing, but it does limit their usefulness.

1.5.4 Selection Process for Labeled Datasets to Use for Tests

As mentioned previously, hate speech datasets appeared to be ideal candidates for sentiment detection. However, they still need to be further filtered based on some criteria. Many of the available datasets which are labeled for hate speech detection tend to have more labels than a simple binary classification (of either hate or not hate speech). Even when they have more than the binary result, the categories are either just a breakdown of the hate speech category by type or the inclusion of an additional "neither" category. The three most significant labeled fields for this research are essentially as follows:

1. Optional: Data source (Twitter, Reddit, Facebook, 4Chan, etc)
2. Message Contents (Dataset text field, comment post, image subtext)
3. Classifier (hate speech, not hate speech, some finer category)

Any and all datasets used for testing models will have to adhere to these basic requirements in order to have consideration. More data is preferred and will be easier to test accuracy with. Depending on the dataset, extrapolating this information has different challenges. For instance, some datasets that are large are "multimodal", meaning they contain classification data that is not exclusively for text but also images or other media (Gomez et al., 2019). These can't be tested against text-only models, so additional work is needed to extract only classified text content from the larger dataset in this case. Ideally a dataset will already have user-identifiable information scrubbed and be stored in a format which is easy to work with for parsing (unusual unicode characters removed, emojis converted to text, et cetera).

Additionally, it is important that the datasets being fed to models for testing aren't comprised of data that the models were originally trained on. To get a good idea for their detection accuracy, it is important that they are being provided with novel data that may not necessarily align with training expectations.

1.5.5 Use of Multi-Model Testing for Bias Reduction

The path of least resistance to deciding what kind of sentiment analysis to perform against datasets is to aggregate multiple machine learning models which already exist for content detection against natural language datasets. Hugging Face, given its large repository, is an example of a good place to look for sentiment analysis models. Many are using base NLP models like BERT and are trained to look for similar sentiment (toxic language, hate speech, etc), but perhaps have differing

training bias. Additionally, any machine learning model(s) employed to perform sentiment analysis are going to include some inherent bias based on whatever the original training data or intent was. There is additional bias from the creator of the model and from the topic of sentiment to be analyzed. Thus, employing multiple models or training models with diverse data can even out the bias and improve accuracy (Gaikwad & Thool, 2015).

From a code standpoint, multiple pre-trained models can be fed the same line of text and then an increasing confidence score can be assigned to an individual message as it passes through each model's "filter" for stronger accuracy. The models can together independently process a single message and derive sentiment from it, adding to the ultimate score. This approach is a more simplified version of what is normally called "Bagging" or "Bootstrap Aggregation" when training machine learning models (Gaikwad & Thool, 2015), except in this case the models will have already been trained and their aggregate results will be taken. This way there can be a more accurate assessment of whether a message fits a classification without the need to rely on a single model and its associated training bias. The following models have been selected based on the previously mentioned positive characteristics.

Model 1: **Roberta Hate Speech Dynabench 4** (Vidgen et al., 2021)

The Roberta Hate Speech Dynabench 4 is a model from Learning from the Worst: Dynamically Generated Datasets to Improve Online Hate Detection (Vidgen et al., 2021) which produced results at only a 27.7% error rate during its trials and was trained on over 40,000 labeled data entries. This makes it a robust candidate to perform preliminary testing against, not to mention it was produced by Facebook AI research, which means it was likely designed with platform use intent in mind. The true test of its capability will be feeding an external dataset and observing its sentiment analysis capabilities outside of its fixed set. This model is fine tuned from the natural language processing framework BERT, as previously discussed (Devlin et al., 2019).

Model 2: **Twitter 2022 154M** (Antypas & Camacho-Collados, 2023), Trained against 154 million tweets for hate speech detection, MIT licensed and the latest version is fine-tuned against 13 more datasets. This model also uses the BERT underlying model for its natural language processing.

Model 3: **198M Multilingual Hate Speech Classifier for Social Media Content** (Barbieri et al., 2022) also trained against a separate independent scrape of 198 million tweets. This model is designed to work with multilingual data, arbitrarily Arabic, Croatian, English, German and

Slovenian are also included. This model was selected for also having the MIT free use license. Also its multilingual training may produce interesting results with sample data. The other characteristic that make this model interesting for testing is that its sentiment output offensive / not_offensive, rather than specifically indicating hate or not hate. It has a set of different qualifiers that it is ranked on scoring for, with hate speech itself only constituting a a portion of the final "hate" ranking for the sentiment, so it should produce interesting performance results.

All three candidate models in the end are selected due to robust training methods and accompanying academic papers to rationalize their methods. There are many models which are available for use, but not all are subject to rigorous scientific process in their production, regardless of accuracy. Additionally, "heavy" sentiment analysis testing will not be performed while the bot is deployed. Sentiment analysis accuracy can be tested independently from the bot itself and then validated as a proof of concept in tandem after the results of the underlying sentiment analysis code are determined.

1.5.6 Calculations to Determine the Performance Accuracy of Model Sentiment

A good statistical method for determining how good a binary classification (positive or negative) sentiment analysis is at detecting something is the F-score. It is called the "harmonic mean" of precision and recall and it works by comparing true positive, true negative, false positive, and false negative binary classification results (Sokolova et al., 2006).

Class \ Recognized	as Positive	as Negative
Positive	tp	fn
Negative	fp	tn

Note: adapted from Beyond accuracy, F-score, and ROC (Sokolova et al., 2006)

The adapted table above helps to illustrate this "confusion matrix"

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

$$F = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The following is a summary of F score Adapted from (Taha & Hanbury, 2015) and used in a variety of works (Cooke et al., 2023; Mody et al., 2023; Sokolova et al., 2006; Taha & Hanbury, 2015; Vidgen et al., 2020)

F-score is calculated based on the 2 values of recall and precision. Recall is the measure of true positive (TP) to false negative (FN).

Recall *is the measure of true positive (TP) to false negative (FN).*
Precision *is the measure of true positive (TP) to false positive (FP).*
F-score *is the "harmonic mean" of precision and recall.*
As F approaches 1, it indicates higher detection accuracy.

Because a baseline for true positive and true negative must first be set and manually determined, this rough precision determination can be done in small subsets of random samples of messages from the dataset and then averaged to get a rough idea for how well the code will perform against unknown data. Many downloadable text classification models online seem to come with a predictive F-score, Precision, and Recall (among others) to indicate their accuracy.

1.5.7 Code Hosting for the Bot

A consideration must be made for where code will reside to achieve both goals of sentiment analysis and bot development. The more important aspect of this workflow is the sentiment analysis, as it is performing the detection. That ultimate goal is more significant than an ultimately functional bot. From a hosting standpoint, the sentiment analysis code can reside anywhere as long as it has access to either libraries and APIs to talk with machine learning frameworks or repositories. In practice, this can be any Linux or Windows PC or server for prototyping and can move elsewhere when it comes time to test the detection code alongside the bot code. For the bot code, it matters a

bit more where it is hosted from a practicality standpoint, as it must be able to speak with the Telegram API using Botfather (Modrzyk, 2019) and have enough security to prevent unwanted access or privilege escalation on the machine that it runs on. For this research, the bot can run simply off of a local laptop.

1.5.8 Bot Security Implications

One of the challenges associated with bots on Telegram is that they are public facing by default and, when deployed, have no inherent security built in to prevent anyone on the platform from interacting with them. Once an API key has been made and given to the code running the bot, registered with the Botfather, anybody with access to Telegram could hypothetically interact with the bot. Thus there are two main considerations that have to be taken into account when making this detection bot for testing purposes.

1. The bot should reside somewhere that, should it prove to be insecure, does not allow access to private data or important resources in its host environment (IE. a personal computer).
2. The bot should include a security feature to prevent regular Telegram users from accessing it and chatting with it. This can be built in the form of some authentication challenge or a simple user whitelist.

Serverless (function-based) infrastructure is one potential option for hosting this bot. It can be done within the computational limits of the free tier of many major cloud providers (*Serverless Computing – AWS Lambda Pricing – Amazon Web Services*, n.d.). If serverless infrastructure is compromised, it does not reside on a monolithic server instance where a potential malicious actor would have access to many other resources aside from the bot. For the purpose of short-term testing, consideration number 2 is really the only one that matters in the end.

1.5.9 Communicating With the Bot Through an Interface

When performing sentiment analysis with basic code, results are simply output to a terminal display. With a bot the communication between sentiment analysis, the actual bot code and the end user is via the interface of a Telegram chat or channel rather than a terminal's output. From a practical standpoint the bot can be deployed in a one-on-one chat where data is fed and the bot responds with sentiment analysis results. This would look something like a call and response where some conversation data is fed to the bot as one message and the bot responds with an interpretation of that message. Another way to deploy the bot can be as a passive observer in a simulated channel,

where it will actively listen in the “background” and all messages into the channel will be processed by the bot and it responds only if a certain prediction threshold is achieved. In the backend, the bot’s code will simply be receiving inputs and responding with results from arguments passed to machine learning model(s). Again, the sentiment analysis will be testable via a CLI, but for the deployment of the bot itself, the Telegram chat will stand as a proxy for a CLI with the bot. Rather than sending commands the bot will be listening for input and responding either asynchronously or after given intervals.

1.6 Conclusions of Chapter 1 – Literature Analysis

Automated content moderation is an essential practice on major social platforms due to scale and multivariate pressures. Exploration of novel independent moderation strategies provides unique insight into these platforms.

Moderation bot should be able to actively monitor simulated conversation in order to be comparable to stand-in realistic human moderation and also be ideally more effective than simply a whitelist or filter. The accuracy of the model performance can be tested independently from the bot.

Python is the preferred language of choice, due to its ease of access and widely available libraries and frameworks for working with both Telegram and AI. Tensorflow will be used to interact with machine learning models and all trained models will be pulled from the HuggingFace repository.

The Natural Language Processing Models used to develop the bot should employ pre-trained model based on a proven framework like BERT. They must also come from a research background and have varying degrees of bias and data used to train them.

The machine learning models will be performing binary classification via sentiment analysis for text based data.

Several large and labeled hate speech datasets from social media content are ideal candidates for testing sentiment against.

2. DESIGN PROPOSAL

From a practicality standpoint, there are two primary goals that this content detection strategy will be attempting to achieve. The first goal will be simply the *detection* of content, (whether truly "hate speech" or not is less significant than the ability to detect the content based on labeling). The secondary goal will be the *application* of this detection method in a trial within its natural environment (Telegram chat, in this case). The rest of the process flow for this research stems, logically, from the needs of both of these goals.

2.1 High Level Workflow for Sentiment Analysis

Figure 2 depicts a high-level overview of the steps for achieving machine learning sentiment analysis.

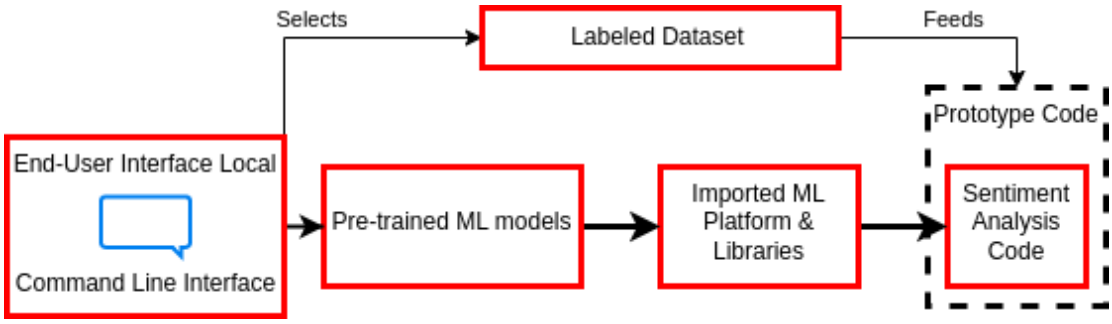

Figure 2: High Level Workflow Diagram for Sentiment Analysis

The end user interface of sentiment analysis testing and functionality is a CLI. The backend code is made up of a pre-trained machine learning model or models that are imported via machine learning platform libraries (such as Transformers). The analysis is a result of the output of feeding in a labeled dataset with deterministic results. This architecture will be further decomposed in its individual approach subsection.

2.2 High Level Workflow for Telegram Bot

Figure 3 a high level overview of the workflow for interfacing with a telegram bot from the chat interface and how it interacts with the backend bot code via the Telegram API.

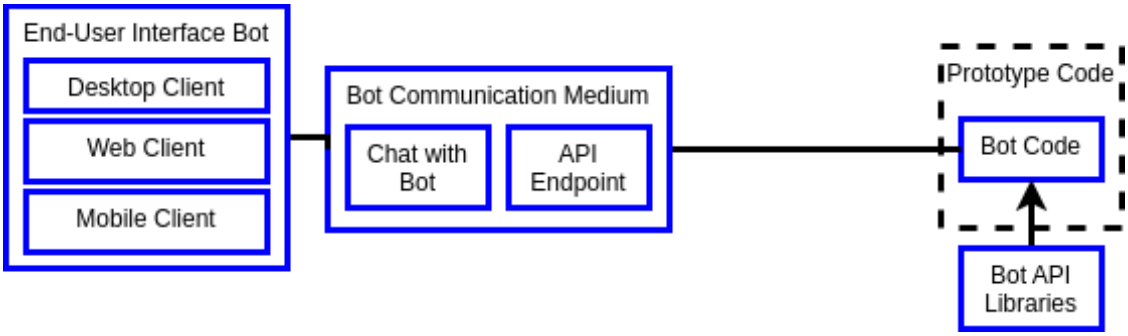

Figure 3: High Level Workflow Diagram for Telegram Bot

The bot code interacts with an API endpoint and authenticates with Telegram using an API key. There are many API libraries for popular languages, but the one that will be used here is for Python because it has robust libraries that support both bot development and work with machine learning models. This workflow will also have further decomposition in its respective approach section.

2.3 Combined High Level Workflow for Sentiment Analysis and Telegram Bot

Figure 4 depicts a combined overview of the two approaches working in tandem with some additional considerations for the hosting location for the respective approaches.

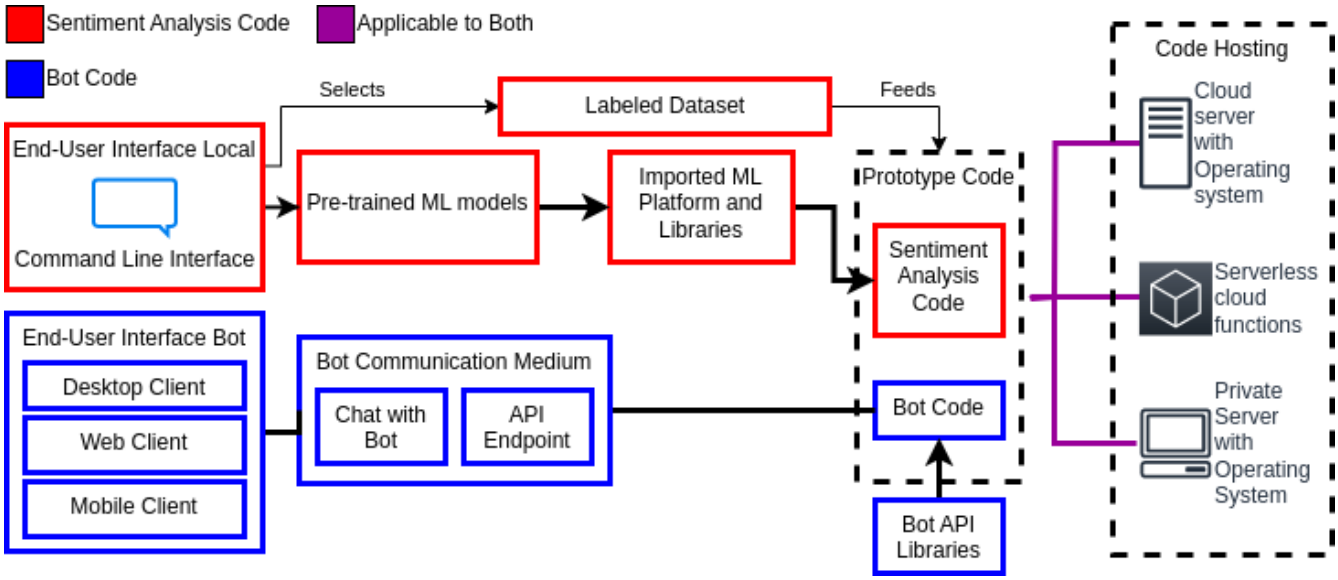

Figure 4: Sentiment analysis code and bot workflow diagram

*Pictured in **red**, code for performing sentiment analysis against a given dataset (to detect something).*
*Pictured in **blue**, code and pipeline for performing bot related functions and interacting with bot itself (to apply the detection strategy in the platform).*
*Pictured in **purple**, considerations that will apply to both bot and sentiment analysis.*

In addition to the combination of both strategies, there is also a small additional section showing the consideration for where code will reside.

2.4 Sentiment Analysis Operational Capability Breakdown

Figure 5 is a diagram of the operational capabilities required for sentiment analysis, roughly corresponding to a use case diagram without system decomposition.

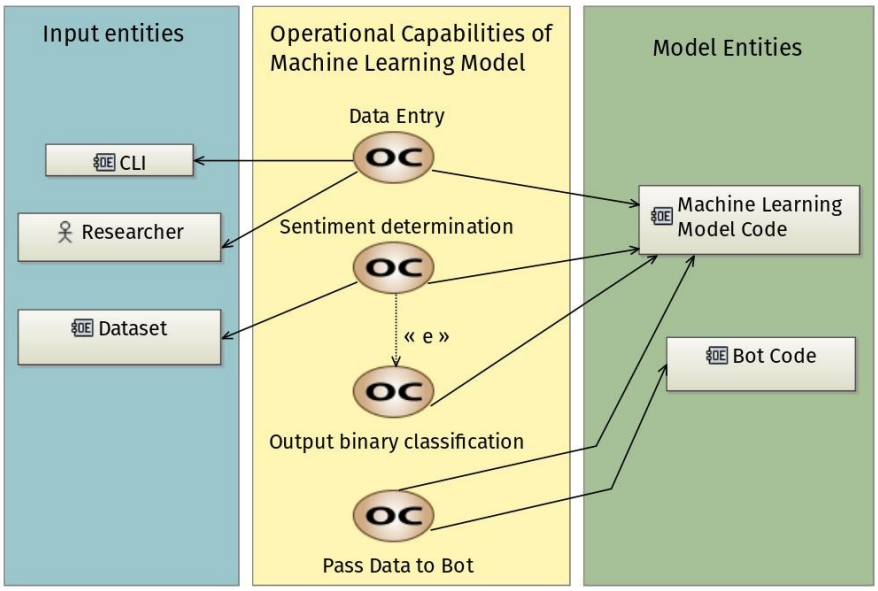

Figure 5: Sentiment analysis operational capabilities diagram

In the center are capabilities the system should achieve and on the outside are entities that participate in achieving those capabilities, be they human or non-human factors.

Table 7: Table of Sentiment Analysis Entities and Descriptions

Entity Name	Associated Capability	Description
CLI	Data Entry	Command Line interface for interacting with machine learning model
Researcher	Data Entry	User feeding data into machine learning model
Dataset (s)	Sentiment Determination	Labeled data being fed into machine learning model(s)
Machine Learning Model Code	Data entry, Sentiment determination, Output Binary Classification, Pass Data to Bot	Code the compromises machine learning model for sentiment analysis
Bot Code	Pass Data to Bot	Code from bot that receives sentiment analysis inputs

This table is a breakdown of the high level entities that shall comprise the sentiment analysis aspect of the design.

Table 8: Sentiment analysis capability names and descriptions

Capability Name	Description
Data Entry	Capability of sentiment analysis model to receive message data an inputs
Sentiment Determination	Aspect of machine learning model that produces a binary output for a given input message
Output binary classification	Capability of sentiment analysis model to produce output yielding true, false, positive, negative sentiments
Pass data to bot	Aspect of code that handles results data being fed to bot for output to user

This table is a breakdown of the capabilities that the system design aims to achieve.

2.5 Sentiment Analysis Requirements Breakdown

Table 9: Sentiment Analysis - Functional & Non-Functional Requirements

Functional	Non-Functional
Use of pretrained ML Model(s)	Predefined Prediction Accuracy / Recall / Precision Determination
Use of labeled datasets	
Use of Bias Reduction and multi-model testing or fine-tuning	

Description and rationale for each requirement from ***Table* 9** included in remainder of ***Section 2.6.***

2.6 Sentiment Analysis Architecture Breakdown

This section breaks down the different steps required for performing sentiment analysis.

2.6.1 Before Sentiment Analysis

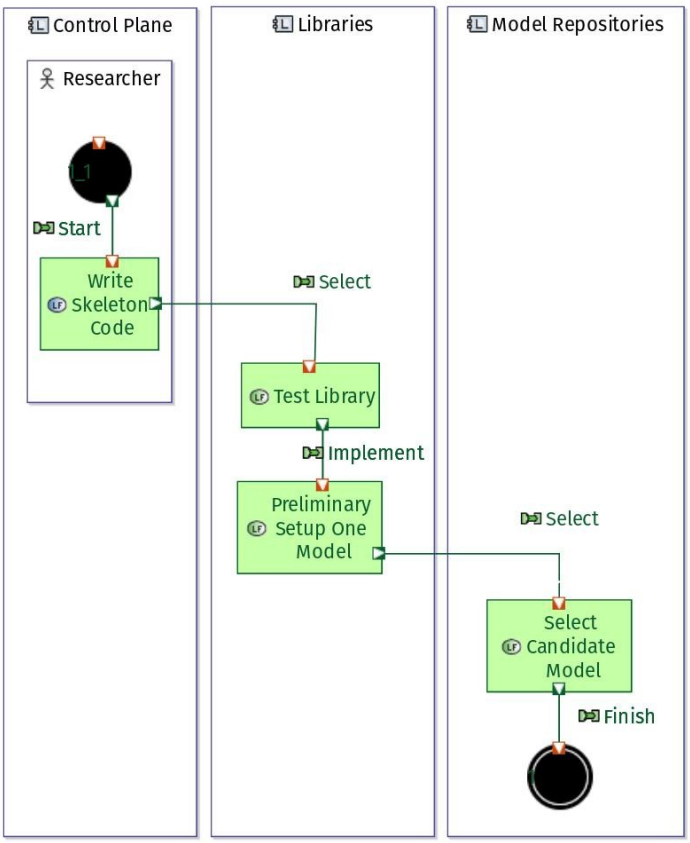

Figure 6: Pre - Sentiment Analysis Activity Diagram

The system needs some preliminary activities in order to perform sentiment analysis. As shown in ***Figure 6*** they are essentially as follows:

- Write skeleton sentiment analysis code
- Include/test library for working with pretrained models (Transformers)
- create an preliminary setup for at least one machine learning model from online repository
- Select candidate pre-trained machine learning models from repository, preferably with topical focus related to content detection

2.6.2 Sentiment Analysis with an Initial Candidate Test Model

The following diagram shows the steps for performing sentiment analysis with a single initial model.

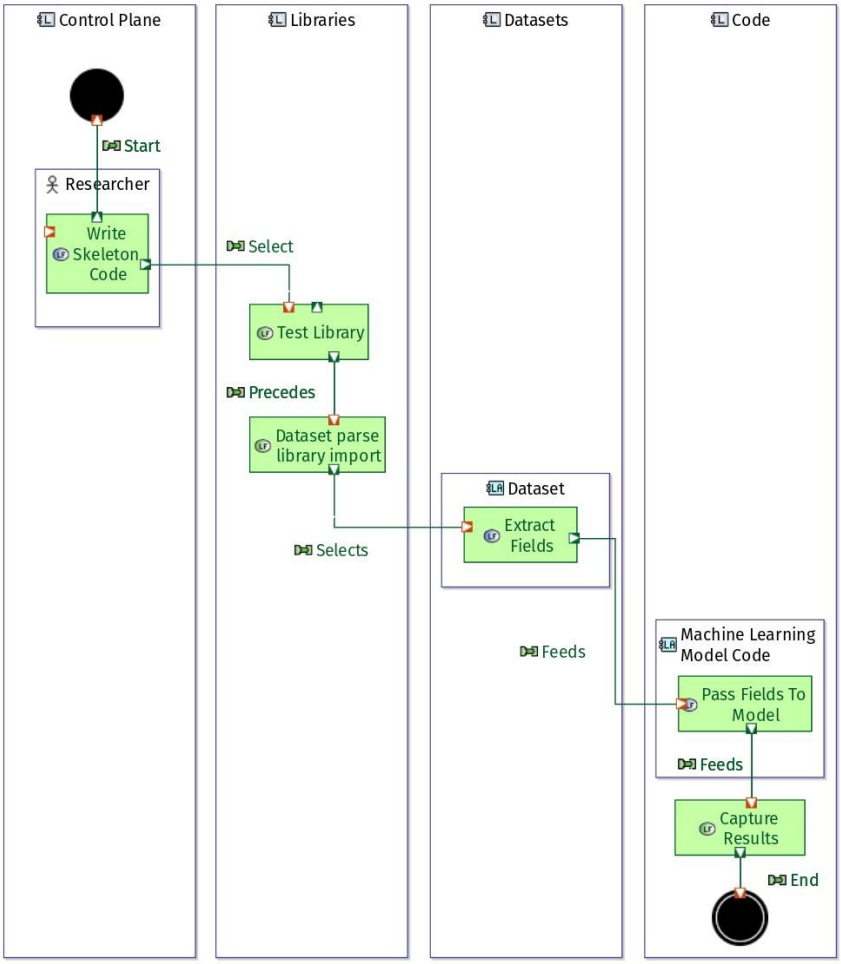

Figure 7: Sentiment Analysis Primary Model Activity Diagram

After some preliminary setup, the rest of the process for the first model is as follows:

- Implement one fully functional machine learning model that can process at least one labeled message at a time from an input field
  - import library for parsing labeled data frames (JSON)
  - import library for importing and passing data to ML model (Transformers)
  - extract relevant fields from labeled dataset (ID, text, sentiment_result)
  - pass fields from dataset into model as input
- Capture primary model outputs and results
- Store results with input data to compare sentiment accuracy

2.6.3 Sentiment Analysis with Additional Models

The process for performing sentiment analysis with additional models is exactly the same as with the first model, only repeated a few more times.

Once an initial model has been tested, the following tasks remain to complete all performance tests:

- Implement additional pre-trained machine learning models from repository (HuggingFace)
- Implement additional code to capture aggregate outputs to produce a single resultant binary classification (IE. hate speech or not hate speech), for use as output from the bot.

2.6.4 Using F-score as a Measure for Performance

Additional once all of the results are collected their outputs need to be verified for accuracy so that F-scores can be calculated. The flow below shows the steps required to verify accuracy.

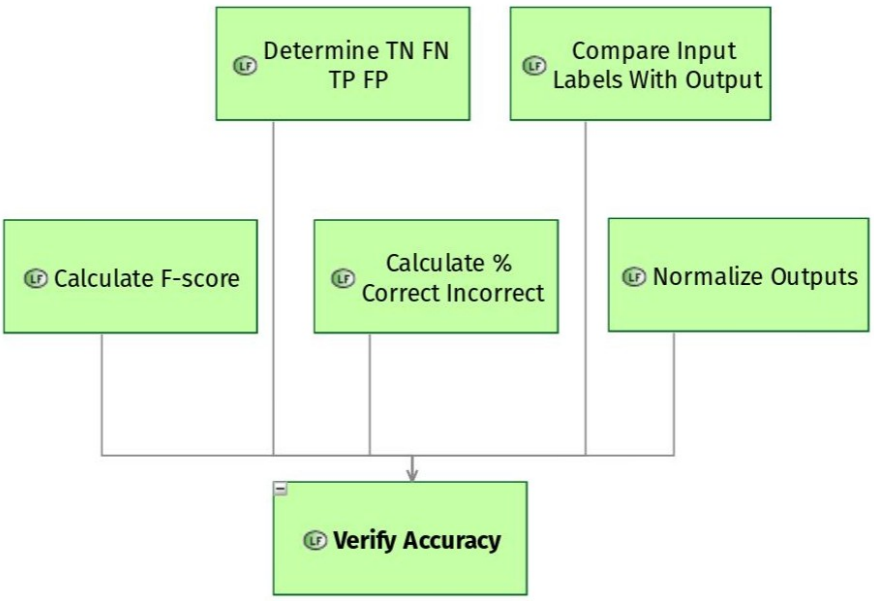

Figure 8: Functional Decomposition of Accuracy Verification Step

Once the initial data has been collected, the accuracy determination can be calculated as follows:

- Perform simple preliminary f-score calculations against initial test dataset with prototype
- The F-score calculation can be decomposed into the following basic parts
    - Normalize model outputs (JSON)

- Compare input labels against output sentiment (Hate Speech true or false)
- Calculate the percentages of incorrect and correct sentiment, based on output labels
- Use the counts of correct and incorrect sentiments to derive TF, TP, TN, FN
- Calculate F-score with those derived ratios

2.7 Telegram Bot Operational Capability Breakdown

Figure 9 is a diagram of the operational capabilities required for creating the Telegram bot.

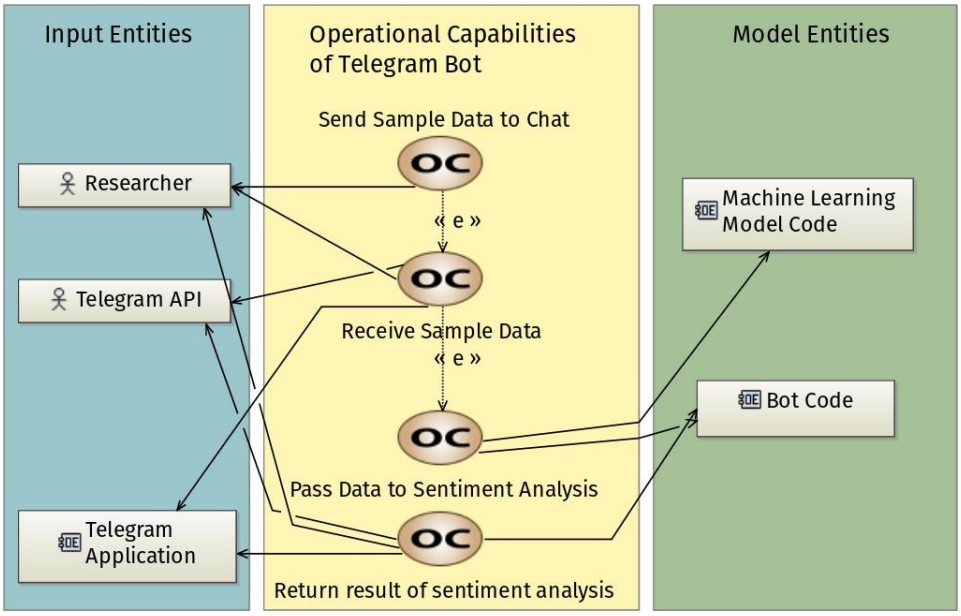

Figure 9: Telegram Bot Operational Capabilities Diagram

This diagram is also roughly corresponding to a UML use case diagram without system decomposition, which will be explored in the next section. In the center are the capabilities that the bot should be able to achieve and on the outside are the entities that will achieve each capability.

Table 10: Table of Telegram Bot Entities and Descriptions

Entity Name	Associated Capability	Description
Researcher	Send Sample Data to Chat, Receive Sample Data, Return result of sentiment analysis	User feeding data into and receiving data from telegram application
Telegram API	Receive Sample Data, return result of sentiment analysis	Programming interface for passing data between researcher and bot code
Telegram Application	Receive Sample Data	User interface for researcher with

		backend
Machine Learning Model Code	Pass Data to Sentiment Analysis	Sentiment analysis code interfacing with bot code
Bot Code	Pass Data to Sentiment Analysis, Return result of sentiment analysis	Code to pass data between API and sentiment analysis

This table is a breakdown of the high level entities that shall comprise the Telegram bot aspect of the design.

Table 11: Telegram Bot capability names and descriptions

Capability Name	Description
Send sample data to chat	Action researcher takes to send data through chat interface
Receive Sample Data	API and bot code capability of handling data passed through interface from researcher
Pass Data to sentiment analysis	Bot passing data from API to sentiment analysis
Return result of sentiment analysis	Sentiment analysis returning result to bot for researcher to view via interface

This table is a breakdown of the capabilities that the Telegram bot design aims to achieve.

2.8 Telegram Bot Requirements Breakdown

Table 12: Telegram Bot - Functional & Non-Functional Requirements

Functional	Non-Functional
Bot Communication Channel and End user Interface with Bot	Code Hosting for Bot
Feeding unlabeled test data and labeling analysis	Bot Code Location and Security Considerations

Description and rationale for each requirement from ***Table 11*** *are* included *in remainder of* ***Section 2.***

2.9 Telegram Bot Architecture Breakdown

This section breaks down the different steps required for building a telegram bot that can perform sentiment analysis.

2.9.1 Bot Registration with API

Figure 10 diagram depicts the steps for registering a Telegram bot for use with the Telegram API.

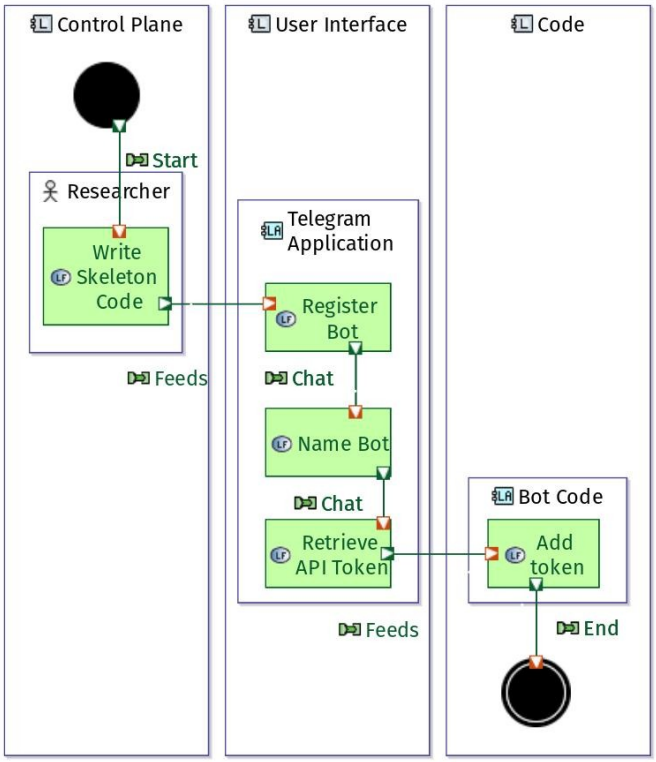

Figure 10: Process to Register a Telegram Bot for Use with the Telegram API

The process for registering a Telegram bot so that it can interact with Telegram's API are roughly as follows:

- Implement some initial bot skeleton code
- Register Telegram bot with Botfather API
  - New bot creation order
  - Name Bot
  - Retrieve API Token
- Add API Token to bot code, test it's ability to communicate inside of Telegram

2.9.2 Bot Testing Before Adding Sentiment Analysis

Figure 11 shows the steps required for getting the bot setup with some minimal features before adding in sentiment analysis.

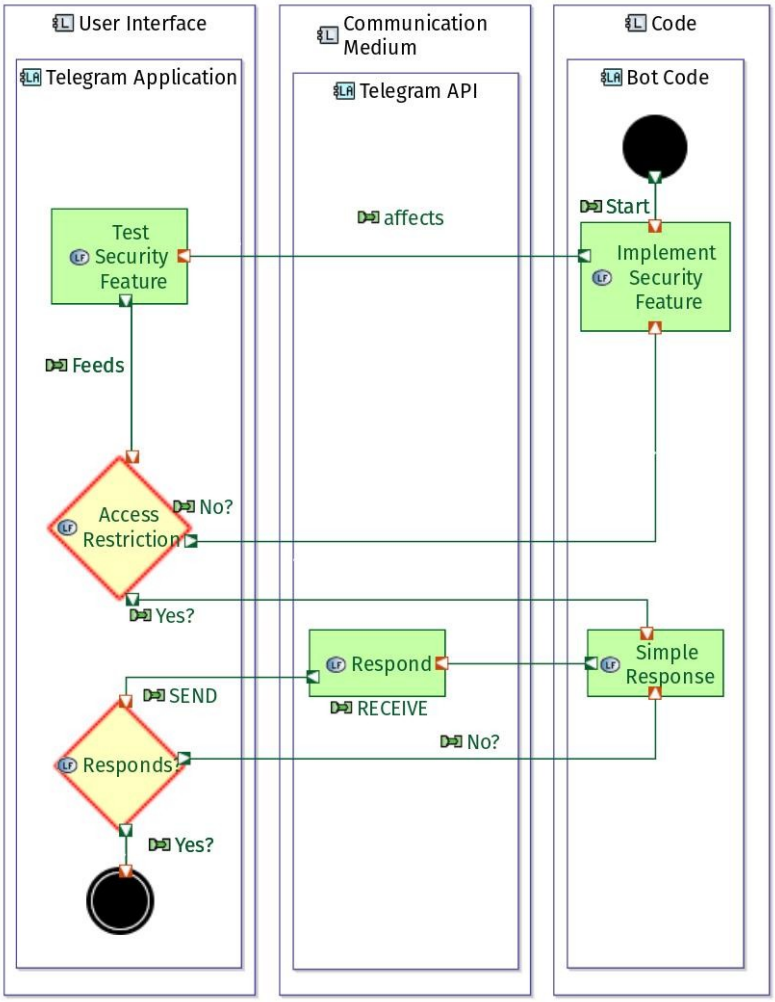

Figure 11: Process for Testing the Bot Before Adding Sentiment Analysis

Simply put, the bot needs to be tested to make sure that it works before adding in the sentiment analysis functionality. Some of these setup steps include:

- Implement security feature in bot to prevent outside access
- Test security feature implementation
  - Can bot be interacted with outside of access restriction?
- Simple output response
  - Test bot call and response to input

2.9.3 Bot Testing With Sentiment Analysis Response

No specific diagram required, since this activity is simply a combination of the previous several diagrams. Its steps are like so:

- Merge sentiment analysis code with bot code
  - Pass sentiment analysis output
  - Send sentiment analysis response via API
- Re-implement sentiment analysis standard input as bot message input
  - Pass message input into telegram
  - Have the bot respond with the result of its sentiment analysis
  - Demonstrate the results

2.9.4 Additional Unlabeled Dataset Analysis

The following diagram shows the steps to perform unlabeled dataset analysis. Though unlabeled data will not be performance tested, it is still a good exercise to observe its complexity.

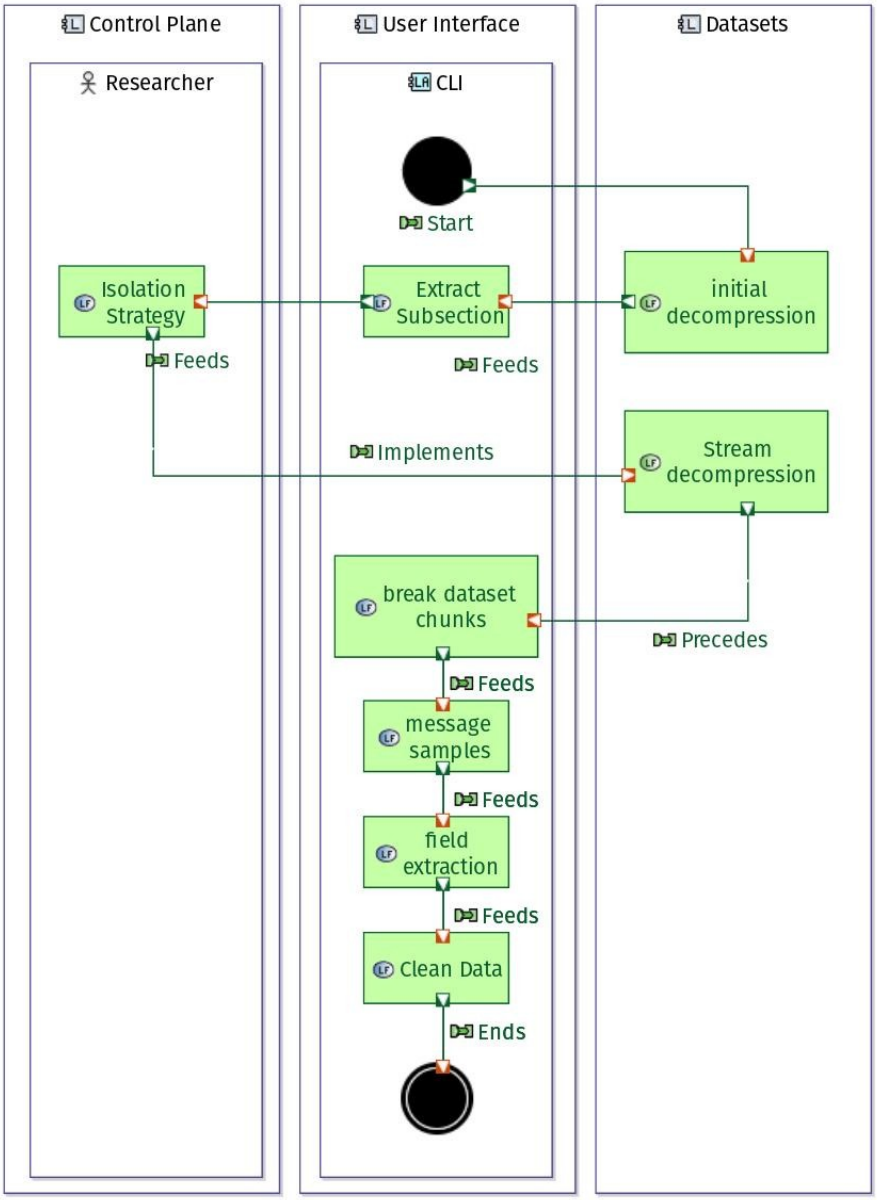

Figure 12: Process for analyzing an unlabeled dataset

Even if it is not required for this research, analyzing unlabeled data is also necessary to understand some of its drawbacks and is included here for additional context around the choice to use labeled sets for measuring performance. The steps are as follows:

- Unarchive small portion of dataset
- Extract small subsection of dataset
- Determine dataset message format and isolation strategy
- Stream decompression
    - Break apart dataset into chunks
    - Extract message samples from chunks

- Programmatically isolate single message field from chunk
- Programmatically isolate multiple message fields from chunks
- Clean unnecessary data

2.10 Conclusions of Chapter 2 – Design Proposal

The design proposal chapter aims to create a framework for the description of the dual-approach for both creation of machine learning model sentiment and a bot that uses that sentiment in practice. This framework is then used to drive the steps of the following phase to completion. It also serves to visually decompose some of the complexity of each individual phase of creating the practical part of this research. As is apparent from the breakdowns, there are two distinct components of this implementation, and real-world examples of content detection follow similar multi-component patterns.

3. IMPLEMENTATION

This section shows all of the steps that went into dataset analysis, performance testing, and building of the content detection bot implementation. Also included is a demonstration of the implementation in practice.

3.1 Dataset Analysis

Before any sentiment analysis determination can be done, it is important to understand why some datasets are and are not easy to work with for performance testing. The following is analysis of the chosen datasets and the rationale for their inclusions based on some useful characteristics.

3.1.1 Labeled Test Dataset Selection and Preparation

Many datasets were considered and only several were found which were in a format that met labeling requirements. Ease of format conversion and lack of use in the training of the candidate models were the biggest selection factors. The following are the three sets which were selected for prototyping and final performance tests.

1. Multi Platform-Based Hate Speech Detection (Cooke et al., 2023)

A small creative commons dataset from the publication Multi Platform-Based Hate Speech Detection (Cooke et al., 2023) was the first used due to its size (3000 messages), ease of formatting, and the use of multiple platforms for data (Reddit, Twitter, and 4chan). Since this dataset is not very large and is relatively recent it has not already been pulled into other larger aggregate datasets and is a good small benchmark to test models against. This set is both for initial prototyping and final performance tests.

2. A curated Dataset for hate speech detection on social media (Mody et al., 2023)

This is a large scale dataset which contains 451709 messages. It is actually already an aggregate set generated from ~20 datasets. There are several quality characteristics in this set that make it suitable for this sentiment analysis. The data inside has been scrubbed to remove any user-identifiable information. Scrubbing, in this case, also includes the exclusion of any model incompatible (escape) characters or unusual line breaks which would be difficult to manipulate with code. The following is a sample from this set, which has a similar format to the Cooke set. Note, that

it does exclude the metadata about the message origin in the final preprocessed set, but this information is present in the researchers' documentation if it is needed.

Figure 13: Curated Dataset for hate speech detection excerpt

```
Thanks very much,0
~SNIP~ invitation as subject for fair comment in the article Leroy w Emery jr,0
Coat of arms I think the coat of arms is wrong Why does it have the UK s cost of arms,0
```

3. A Benchmark Dataset for Learning to Intervene in Online Hate Speech (Qian et al., 2019)

This dataset was originally designed to be used alongside strategies for warning against hate speech interactively. It is aggregated from both Reddit and the social media platform Gab and contains 22,324 comments (Qian et al., 2019). Its labeling was performed by crowd sourcing using Amazon's mechanical turk, so it has somewhat different sentiment than other labeled sets. The version of this dataset being used is also provided by the Mody et al team as part of some of the original data that was used to test the accuracy of their curated dataset models. The data in this set is not part of the larger curated set, so it can also be easily used for testing without overlap. The set is also labeled in the convenient aforementioned format, so it is easy to manipulate for parsing.

Other datasets that were considered, but not used

The Hate Speech Detection Set (MacAvaney et al., 2019) was also considered. However, in its git repository, the data is not normalized and is split between a combination of individual raw text files and a CSV-to-file labeling map.

It should be noted that these different datasets are also selected because of the different methods used to perform the labeling of their "hate speech" data. Different papers have differing opinions on how that labeling should be done. Spreading out the data should also help reduce the labeling bias. The differing rationale and justification for labeling is also a good test against the inherent bias of model sentiment.

As was discussed in the literature review, the primary candidate model which was selected (Roberta Dynabench) was fine-tuned in rounds of around 10,000 messages. So, in order to keep in line with that standard, the total number of tested messages should be at or above 10,000 messages minimum, which these datasets easily satisfy.

3.1.2 Labeled dataset analysis

Labeled datasets, in general, come with various types of metadata depending on the content of the set and the original research intent of the set. The following is an example of a dataset which is difficult to work with for simple binary classification for several reasons (the fact that it wasn't designed with this in mind being among them).

```
"id"    "annot1"    "annot2"    "expert"    "text"  "text.clean"    "annot1.name"   "annot2.name"
"expert.name"   "target.annot1.clean"   "target.annot2.clean"   "hostile.threatening"
"hostile.dehumanization"    "hostile.interpersonal" "COVID relevant"    "EA relevant"   "hashtags.annotator1"
   "hashtags.annotator2"   "hashtags.decision" "East Asia" "China" "Hong Kong" "Japan" "Korea" "Singapore"
"Taiwan"

"idstr_1213452156251987973" "none_of_the_above" "none_of_the_above" "none_of_the_above" "Afraid. "
"afraid.  #HASHTAG" "annotator_vDe7GN0NrL"  "annotator_HtRmsP3KiK"  "expert_CAgNlUizNm" NA  NA  NA  NA  NA
"no_no" "no_no" "hashtags_not_used_at_all_to_identify_themes"   "hashtags_not_used_at_all_to_identify_themes"
   "agree" NA  NA  NA  NA  NA  NA  NA
```

Figure 14: Example of labeled dataset fields

Note how it includes a large amount of extraneous metadata which must be filtered. It does not include binary sentiment labeling, and the relevant fields are interspersed with the actual message content.

By contrast, the contents of the Cooke dataset are single line, comma separated, and normalized in a way that is very easy to work with for parsing.

```
Platform,Comment,Hateful
Reddit, **** I thought they had strict gun laws in Germany,0
Reddit,"I dont care about what it stands for or anything its connected to, I like the shields ...",0
Reddit,It's not a group it's an idea lol,0
Reddit,So it's not just America!,0
```

Figure 15: Cooke dataset good labeling example

Due to the fact that dataset lines could contain many string escaping or unicode characters, it can be very difficult to parse messages without encountering errors related to these oddities. Datasets like this Cooke set are easy to convert to JSON, which allows for easier results comparison for complex strings and manipulation with Python libraries. This is due to the encoding of JSON having good handling for complex strings. After messages undergo JSON manipulation, a map which looks like the following is the output.

```
{
  "Platform": "Reddit",
  "Comment": "**** I thought they had strict gun laws in Germany",
  "Hateful": "0"
},
```

Figure 16: Normalized labeled message with JSON

From here it is much easier to write code to extract single messages or compare results from models.

3.1.3 Unlabeled dataset analysis

The Pushshift Telegram dataset was also analyzed in order to understand the scope of a very large dataset which has not had pre-labeling applied and what it would technically entail to extract message content from it and why it would be difficult to work with. The following numbered steps are what was required to analyze even a small part of the unlabeled data.

1.) Determine dataset message format and isolation strategy

In order to get a small sample of data to begin creating a more robust message isolation script the Pushshift dataset archive decompression can simply be interrupted after a few seconds of extraction initially. The dataset comes with two compressed ZST archives: one archive for Telegram channel metadata and one archive for messages and their associated metadata.

Looking at a piece of the Pushshift dataset, the decompressed format of the data is NDJSON (a line-free compressed version of JSON). A single unit of data and all associated metadata in unformatted NDJSON for one message looks like the following.


```
{"_":"Message","date":"2019-09-08T18:23:10+00:00","edit_date":null,"entities":
[],"from_id":600675780,"from_scheduled":false,"fwd_from":null,"grouped_id":null,"id":23801,"
legacy":false,"media":null,"media_unread":false,"mentioned":false,"message":"Hes in the new
season","out":false,"post":false,"post_author":null,"reply_markup":null,"reply_to_msg_id":23
799,"retrieved_utc":1567978269,"silent":false,"to_id":
{"_":"PeerChannel","channel_id":1271719213},"via_bot_id":null,"views":null}
```


Figure 17: Pushshift Telegram dataset NDJSON string

The only piece of this NDJSON string that matters for sentiment analysis with pre-trained machine learning models is the “message” field.


```
"message":"Hes in the new season"
```


Figure 18: Single isolated message from Pushshift NDJSON string

Thus, there will need to be some code to programmatically isolate messages from their NDJSON strings. This is, fortunately, not too difficult as NDJSON (or rather JSON) is a standard

format for data processing and there are many utilities available to extract individual fields from JSON, such as “jq” on Linux. A script can take the messages as input and output a file with single lines of every message input field. Some pseudocode as an example:

```
input_file=File_With_Metadata
output_file=File_Extracted_Messages

Remove old_output_file                    # Remove output file if it already exists
While not end:                            # Read the input file line by line
      Read input_file                     # Extract the "message" field
      Message extract regex               # Check if the message field is not empty, and write
      Write output_file
```

Figure 19: Pseudocode for message extraction

A script like this can parse out messages and feed them onto individual lines in an output file such that they are just the message field. The following is an example of what a conversation from the dataset being extracted looks like.

Wow that's Colgate
oh, just a realignment?
never extract ur teeth
did you get your teeth extracted?
Always wear your retainer
I was to lazy wear retainers and I regret it
Even less expensive
thanks for the advice
Remember to wear your retainer

Figure 20: Example newline isolated messages

2.) Break apart dataset into sub-archives

Because the Pushshift Telegram archive is so highly compressed, trying to extract the entire JSON text onto a local hard drive is both unfeasible and unnecessary for data analysis. What makes far more sense is to break the large archive up into smaller pieces to decompress message data from each piece. The ZST archive format has libraries to support “stream decompression” (*Decompression APIs — Python-Zstandard 0.22.0-Pre Documentation*, n.d.). Essentially the archive can be decompressed in sections only without writing the entire decompressed dataset to a disk.

3.) Extract message samples from chunks

In a decompression loop, 1 in every Nth JSON string can be extracted and stored. In this way radically smaller sub-datasets can be extracted from the larger dataset and analyzed. Thus, say, 1 in every 100,000 messages can be isolated and stored to produce a small dataset for testing. Some pseudocode for how that works looks like:

```
 input_file=Pushshift_JSON
 output_file=Extracted_JSON_Lines
With steam_reader                                # Initialize ZST stream reader
       chunk = Reader.read                       # Storing a specified chunk size
       data = chunk.decode #With UTF-8   # Decoding the chunk
       object = JSON.decoded             # Storing decoded chunk in an object (and iterating)
       output_file.write(decoded JSON)   # Writing decoded chunk to a file
output_file.close                              # Save and close the file
```

Figure 21: Pseudocode for Nth string extraction process

In testing this, there are some issues with Unicode encoding and malformed JSON depending on where in a chunk the archive is broken up. This can be accounted for by storing an error ratio against the expected message output. Not all of the data need be extracted and some of the data would have to be culled due to the dataset not containing only English language text or non-ASCII characters. Some of the text is also not conversational, but rather contains links and other miscellaneous non-natural language content. If this dataset is indeed used to test against models, some amount of manual dataset review and classification will need to be performed before machine learning model tests. There are entire academic papers dedicated to properly processing a dataset like this, so unbiased and accurate labeling is unfeasible for the scope of this research.

3.2 Sentiment Analysis

Once datasets have been analyzed the code implementation that is used to feed data for sentiment analysis can be created.

3.2.1 Writing Skeleton Sentiment Analysis Code

In order to work with models which are from HuggingFace there are a few different libraries that can be used. Transformers or PyTorch are some of the more common libraries. Transformers can be used to interact with PyTorch models as well. It is suited for models that are downloaded from HuggingFace as they have well-written documentation on how to work with the library. The code for this initial setup is as follows:

1. Import transformers and torch libraries
2. AutoModelForSequenceClassification retrieves model from repository
3. AutoTokenizer automatically generates class with structure that weights can parse
4. Collect data (in an array, or rather, batches of arrays)
5. Pass data into tokenizer to give it a meaningful form for the model
6. Pass tokenized data to model input and capture sentiment output somewhere (to file)

3.2.2 Implementation of One Fully Functional Pre-trained Model

From the forms of prohibited content on media platforms and a review of pre-existing models which are available in online repositories, the decision was made to use models which are fine-tuned for the detection of hate speech specifically for three reasons:

1. There are many models designed for this purpose.
2. Hate speech is prohibited on many social media platforms
3. There are also many hate speech datasets against which to test these pre-existing models.

The Roberta Hate Speech Dynabench 4 is authorized according to its Arxiv publication to be repurposed for research. The process of implementation of the model is fairly simple and requires the Pytorch and Transformers libraries. The more challenging aspects of working with it are: feeding extraneous data, using a format which is correctly tokenized, and having the model's sentiment be recorded in a way that is easy to manipulate.

The Transformers library supports a class which can “auto-tokenize” input data to a machine learning model. This simply means that data can be structured in a way that the model’s weights will be able to interpret it. The inputs can be tokenized by feeding an array of input strings like so.

```
tokenized_inputs = tokenizer(dataset_array, return_tensors='pt', padding=True,
truncation=True)
```

Figure 22: example function for passing data to initialized model

Initially the idea was to pass data lines JSON formatted in one at a time to observe the model’s behavior and sentiment response capabilities.

```
└─[☕]  git:(main) ✗ head HateSpeechDetection_messages.txt
[redacted] I thought they had strict gun laws in Germany
I dont care about what it stands for or anything its connected to, I like the shields ...
It's not a group it's an idea lol
So it's not just America!
└─[☕]  git:(main) ✗ ./feed_model.sh HateSpeechDetection_messages.txt
input file is HateSpeechDetection_messages.txt
[
  {
    "Comment": "[redacted] I thought they had strict gun laws in Germany",
    "Hateful": "0"
  }
]
[
  {
    "Comment": "I dont care about what it stands for or anything its connected to, I like the shields ...",
    "Hateful": "0"
  }
]
[
  {
    "Comment": "It's not a group it's an idea lol",
    "Hateful": "0"
  }
]
```

Figure 23: Line by line sentiment analysis using example dataset

As can be seen from the standard output, this worked well to demonstrate the model’s functionality, but it was painfully slow (around 1 message per second). The model input parsing logic was reworked in order to parse an N number of lines of input at a time to produce results as quickly as possible. Using batch processing the model could take raw line separated input text and produce sentiment results for 3000 lines in about 1 minute. Some separate code was written both to normalize the input dataset’s JSON formatting and to compare the resulting sentiment against the input.

```
└─[☕]  git:(main) ✗ python3 compare_results.py HateSpeechDetection.json results.txt
Number of Incorrect Sentiments: 204
Percentage of Incorrect Sentiment: 6.80%
View which dict pairs are incorrect in: incorrect_pairs.json
```

The candidate pre-trained Roberta model yielded a **6.8%** sentiment incorrect labeling against the input data’s labels, indicating a good fit for a dataset that it was not trained on. From here the

tests simply scale up the size and diversity of the input dataset and the number of models interpreting the input data. All of the performance results for all datasets can be seen in ***Chapter 4***.

3.2.3 Capturing Some Initial Prototype Outputs and Performance Results

The comparison script was altered to produce calculation of the 4 essential F-score quadrants which can then be extrapolated to calculate any essential statistics necessary.

```
Number of Correct Sentiments: 2796
Number of Incorrect Sentiments: 204
Percentage of Incorrect Sentiment: 6.80%
Count of false positives: 101
Count of false negatives: 103
Percentage of False Positives: 3.37%
Percentage of False Negatives: 3.43%
```

Figure 24: comparison script test result output data for Facebook Roberta

Figure 24 is an adjusted example produced by the input / output comparison script now including all of the required F-score calculation metrics. So, in this example the F-score would be calculated as follows, using the formulas from ***section 1.5.6***

$$Recall = TP / TP + FN = 2796 / 2796 + 103$$

$$= \mathbf{0.9645}$$

$$Precision = TP / TP + FP = 2796 / 2796 + 101$$

$$= \mathbf{0.9651}$$

$$F = 2 * (Precision * Recall) / (Precision + Recall)$$

$$= 2 * (0.9645 * 0.9651) / (0.9645 + 0.9651)$$

$$= 2 * (0.93083895 / 1.9296)$$

$$= \mathbf{0.9648}$$

This is a really promising initial F-score, but it's important to do more rigorous testing to see more realistic results in line with the model's reported performance metrics (27.7%, see (Vidgen et al., 2021)).

3.3 Bot Development

After sentiment analysis, a bot was developed using the python-telegram-bot library in order to pass messages to the models and return detection results.

3.3.1 Bot API registration

Registering a new telegram bot is a fairly simple process.

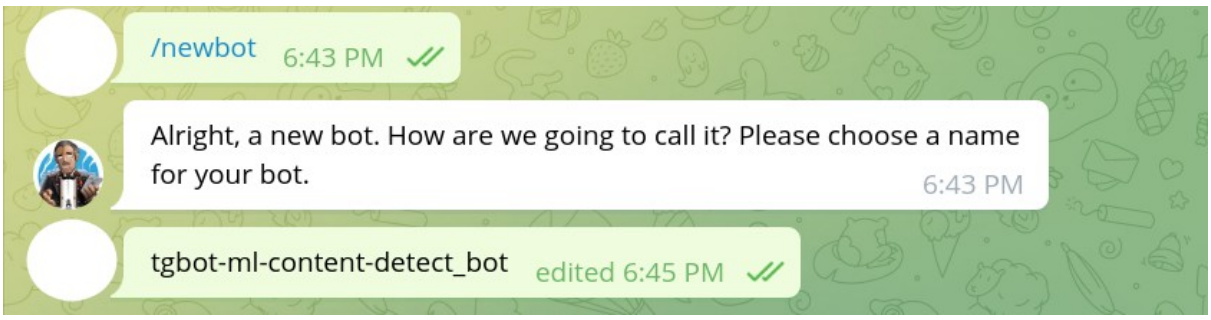

Figure 25: Registering a new Telegram bot with the Botfather

The first registration step in ***Figure 25*** is to simply send a message to the Botfather with "/newbot".

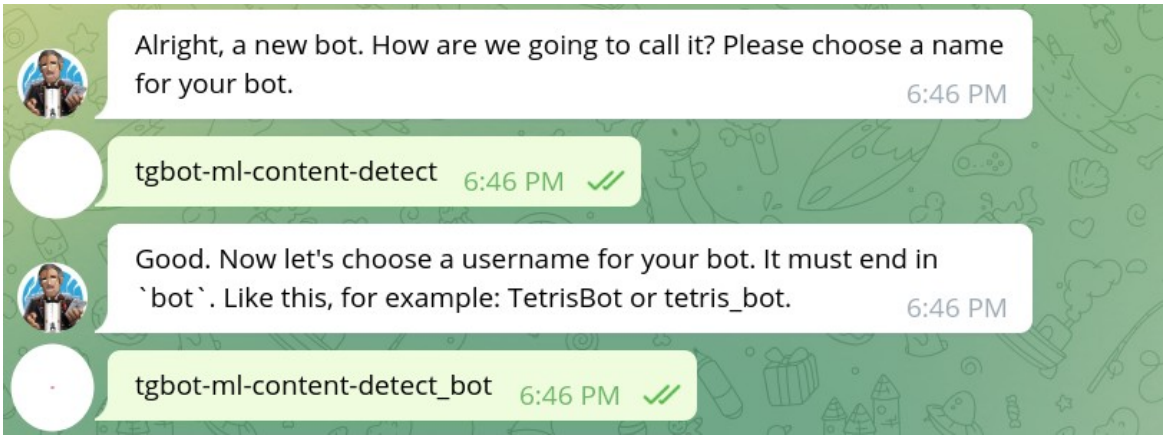

Figure 26: Choosing a name for the new Telegram bot

Then it will ask a few more questions about how the bot should be named (***Figure 26***). Once suitable values are selected, the user is presented with an API token, which can then be used by any program to authenticate in place of the bot. With this token it is possible to read, write, and check

messages and user information. From there, all that's needed is the logic for parsing sentiment from the machine learning models.

3.3.2 Testing the Bot Before Adding Sentiment Analysis Features

The code for the bot's functionality is essentially as shown in ***Figures 10 and 11*** There are some additional functions that are added in order to achieve the rudimentary goals of security and access.

Figure 27: Picture of bot user identity function and sleep status reporting

Figure 27 shows functions for checking the current ID of the user, stop command, and feedback about the bot's current status.

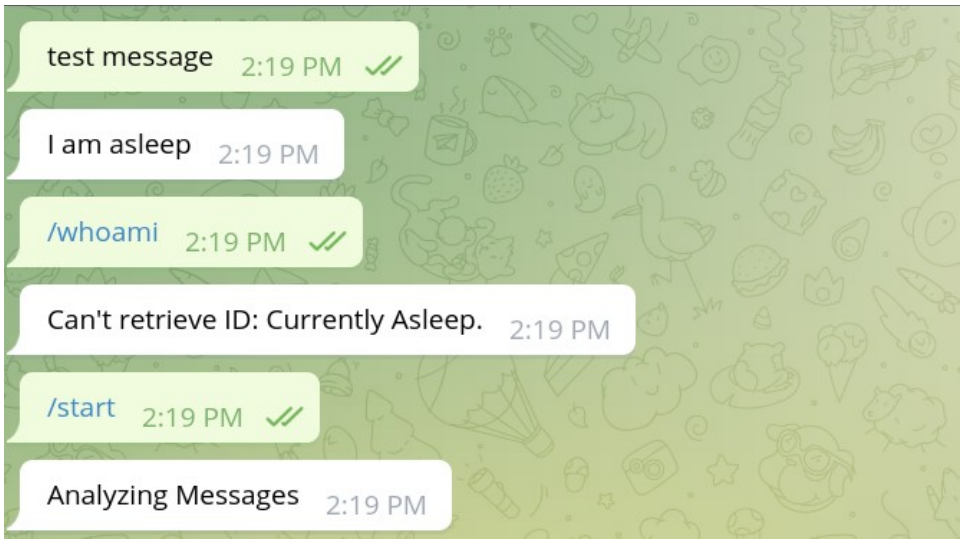

Figure 28: Picture of simple bot sleep status and start function

Figure 28 shows attempting to retrieve user ID while the bot is asleep and then the start command for the beginning of message analysis.

```
def requires_authorization(func):
    @wraps(func)
    async def wrapper(update: Update, context: ContextTypes.DEFAULT_TYPE):
        user_id = str(update.effective_user.id)
        if user_id not in ALLOWED_USERS:
            await update.message.reply_text(str(user_id) + ': Unauthorized')
            return
        return await func(update, context)
    return wrapper
```

Figure 29: Example code showing bot's simple security mechanism

All of the bot's functions are protected by this rudimentary user ID check (***Figure 29***) by using the function decorator `@requires_authorization` such that all asynchronous functions will not run without passing the security check. In the event that an unauthorized user ID tries to interact with the bot, it will simply appear non-functional or, in this case, return an authorization failure. This is a simple example, but the functionality can easily be extrapolated to include multiple user IDs or Telegram channel IDs.

3.4 Implementation of the Sentiment Analysis in Telegram bot

After the initial bot testing for functionality and security was performed, the sentiment analysis code was combined with the bot code so that the bot could perform detection tasks.

3.4.1 Demonstration of Bot Interaction – Single Model Implemented

The figure below is a full example showing off all of the steps of interacting with the bot from start to finish in this sequence:

1. Trying to interact with the bot while it is asleep, showing it's sleep message
2. Trying to retrieve the user ID while the bot is asleep, showing sleep message
3. Running the bot's startup function so that it begins polling for new messages
4. Sending a sample message for the bot to test sentiment against the ML model
5. Returning the raw JSON object with the captured sentiment from the model
6. Trying to retrieve the user ID while the bot is awake, receiving the captured ID
7. Running the bot's stop function, and then testing to make sure it has fully stopped.

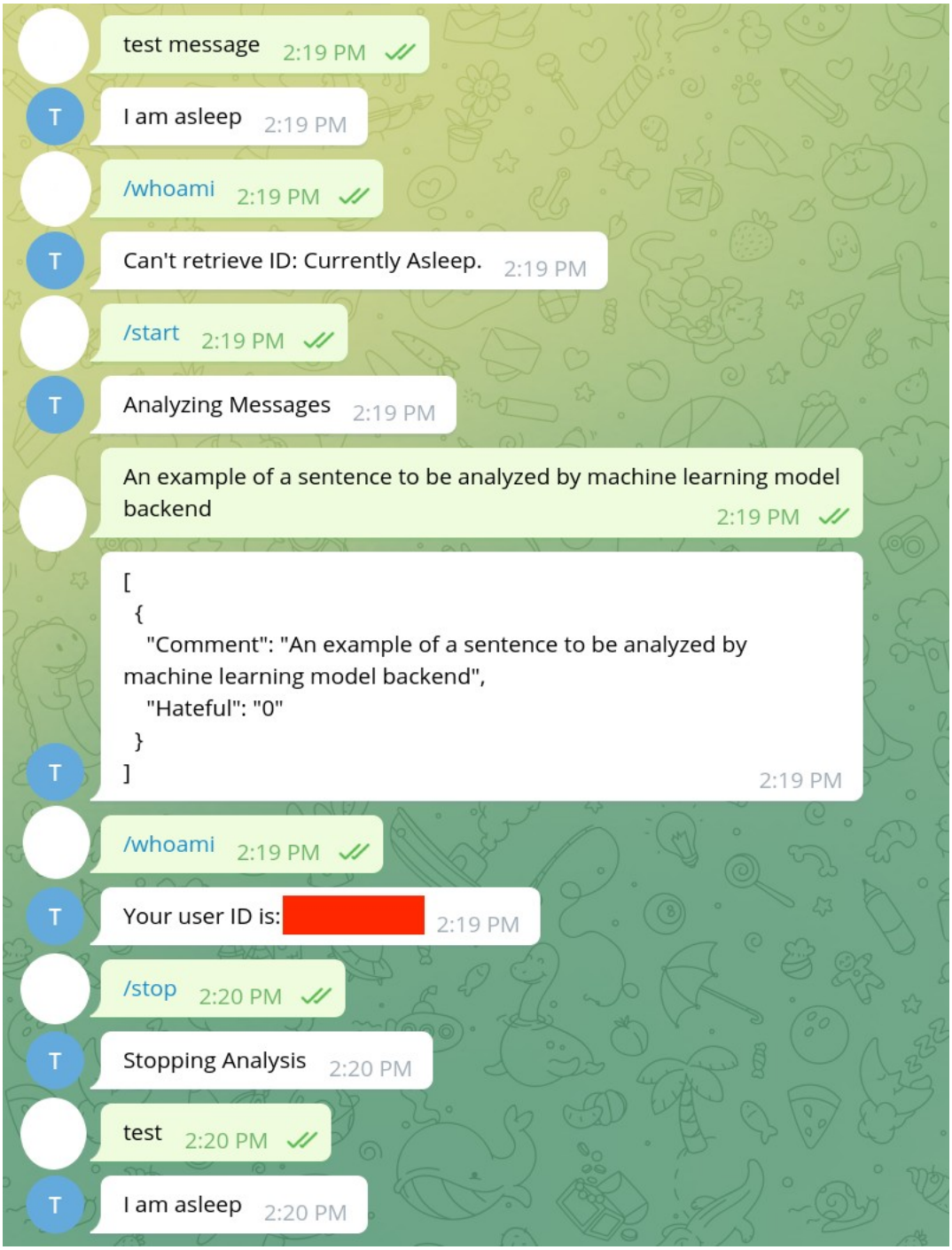

Figure 30: Picture of full bot interaction cycle

The raw JSON of the sentiment is returned to the user for demonstrative purposes, but a productive version of this bot would quietly await messages that trigger sentiment 'Hateful: "1"', before performing a followup action. It is trivial upon calculating the violating sentiment to send a message, for instance, to a specified "admin" user or to locally record a log of the offense for further review. It should also be noted that sentiment response time is roughly around 1 to 2 seconds for a single model.

At this stage some additional tests with filler text were performed to see how the bot responds to maximum message length limits for Telegram.

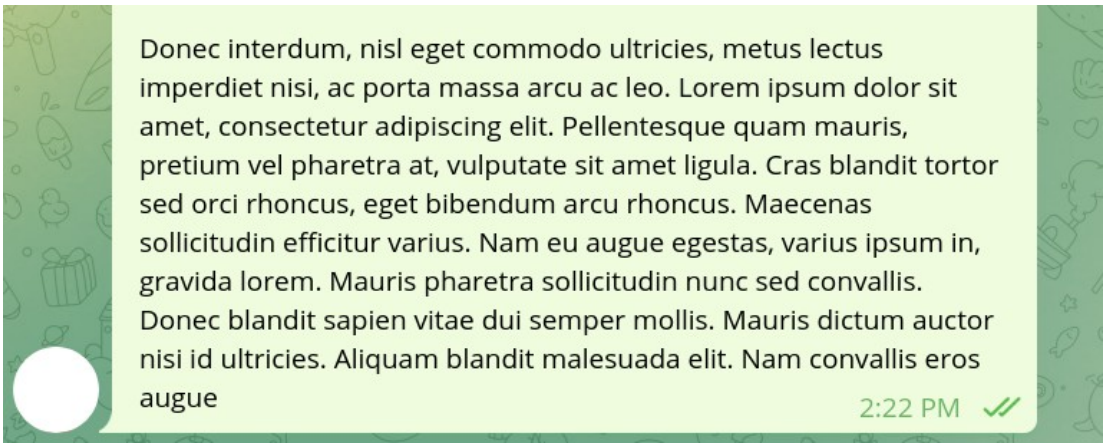

Figure 31: Picture of filler text lorem ipsum message length test

This example message contained the maximum number of characters in a message, 4096 at the time of writing.

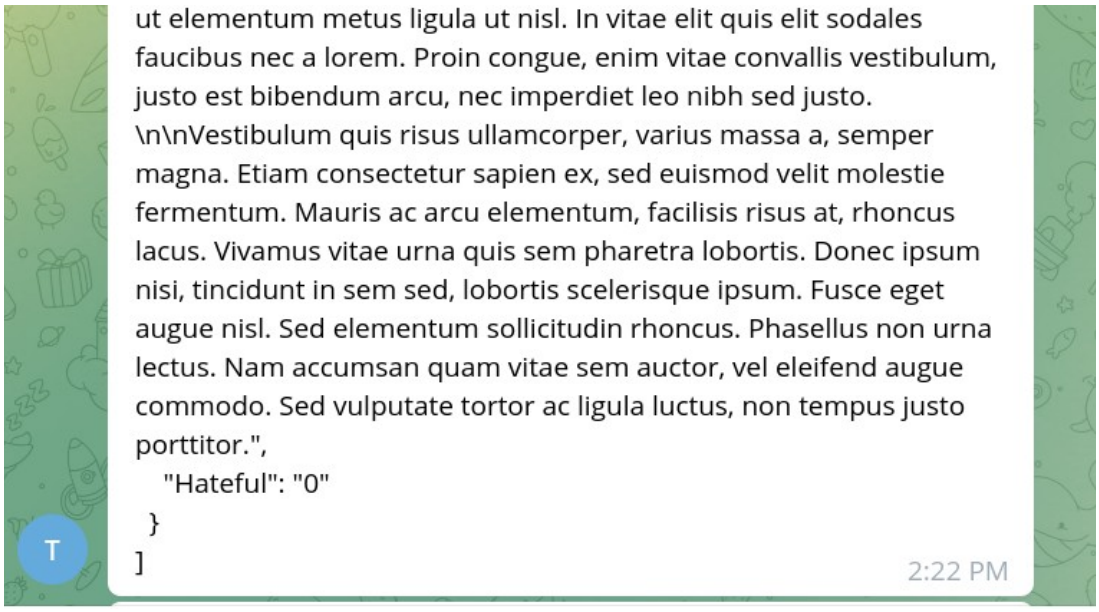

Figure 32: Picture of bot processing lorem ipsum filler message maximum length without issue

The bot simply handled this without issue. Since the length of this Telegram maximum is significantly shorter than the length of some large single messages that were contained in the test datasets.

3.4.2 Demonstration of Bot Interaction – Multiple Models With Aggregate Score

The scores from the 3 models are averaged per message to achieve an aggregate score (using the aforementioned bootstrap aggregation) which would ideally be considered less biased against the sample dataset than a single model.

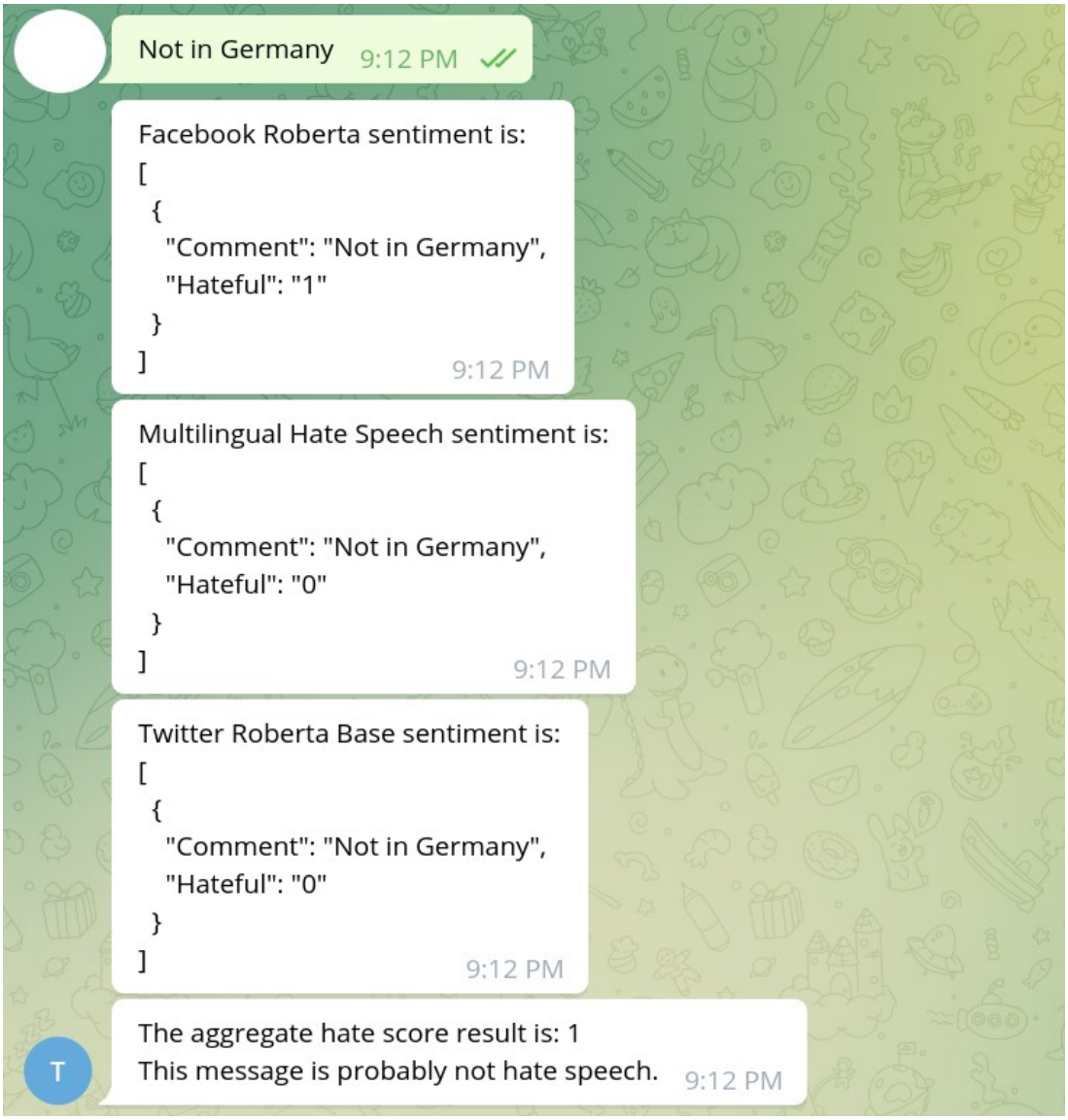

Figure 33: Image of bot returning bootstrap aggregate sentiment result for not hate speech

To demonstrate that it is possible to implement the bootstrap aggregate score, but also potentially useful for true sentiment determination, the bot's code was modified slightly to yield the sentiment from all 3 test models and print the aggregate score to the user along with a message about hate speech likelihood. Once the sentiment derivation was increased to 3 models, there was a slight noted increase in time for response from about 1-2 seconds to around 2-4 seconds per response.

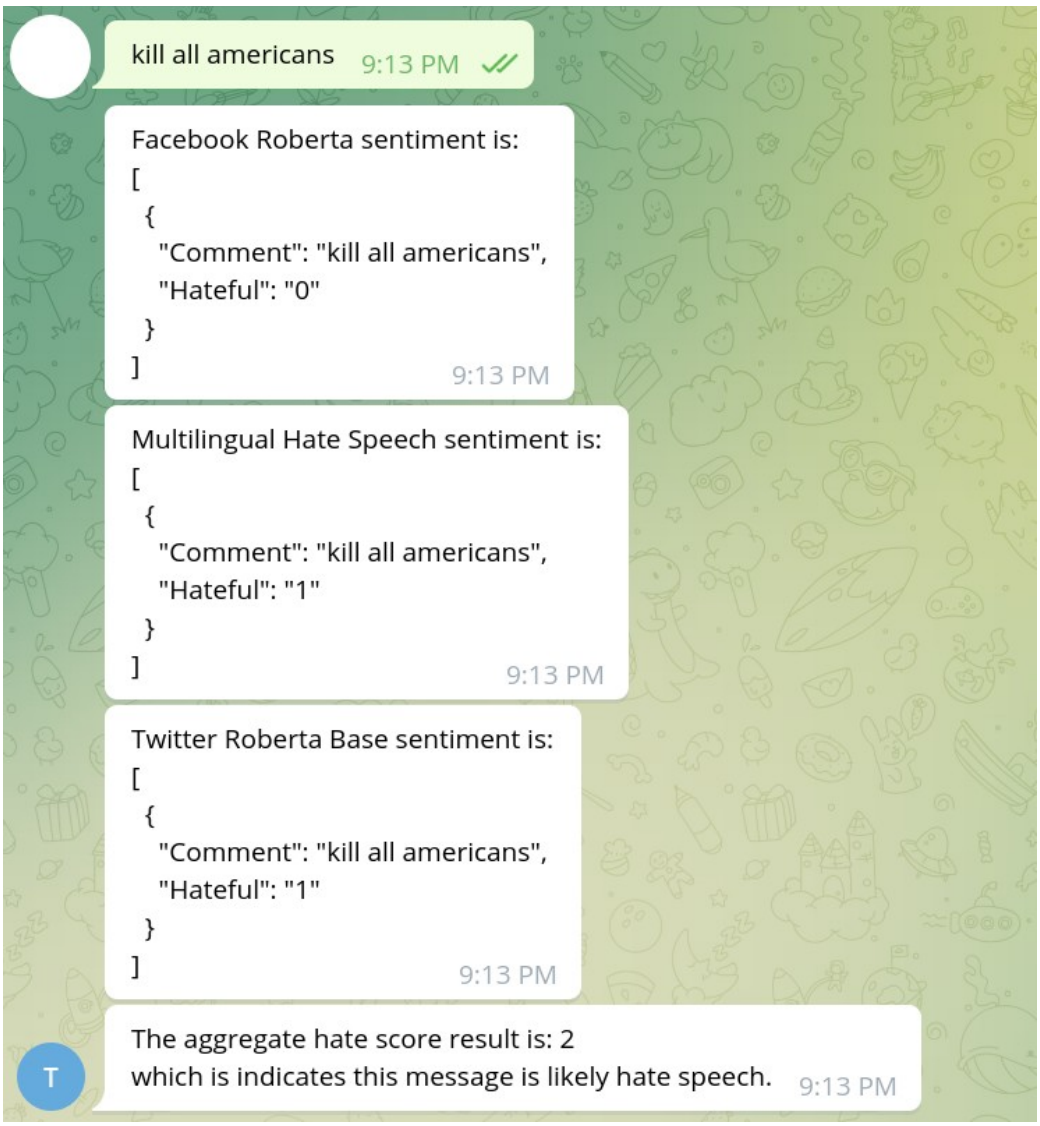

Figure 34: Image of bot returning bootstrap aggregate result for probable hate speech

It was decided that an aggregate hate score of below 2 would be interpreted as “likely” not hate speech, since only 1 or 0 models will have rated it as such. An aggregate hate score of 2 or more indicates that at least 2 out of 3 models interpret the message as hate speech and it is worthy of either action or manual human review.

3.5 Conclusions of Chapter 3 - Implementation

The system design has clear and distinct parts. One for analyzing sentiment, and another for the development of a bot. After creating the breakdown of the architecture and requirements, the difficult section of the work lies mainly in the dataset analysis and testing of machine learning model performance. The breakdown of both labeled and unlabeled datasets showed the value of choosing labeled sets and made the performance tests in ***Chapter 4*** much easier to perform.

Another interesting characteristic that can be seen from ***Figure 34*** individual models perceive slightly different interpretations of individual messages at times and have different bias associated with their training data about what constitutes hate speech. It's clear that using multiple models can potentially uncover sentiment from content that a single model would otherwise miss. In the example in ***Figure 33,*** Facebook's Roberta with otherwise high performance (see ***Chapter 4***) missed a message that would likely be labeled non hate speech by a human reviewer.

4. PERFORMANCE RESULTS

In order to get a good bias spread, the 2 additional candidate models that are also open for research and trained on different hate speech detection data were selected during literature analysis from the HuggingFace repository. The code used to feed these additional models is almost identical to the code for the first model, as they are all being interfaced with through the Python Transformers library. The results of their performance analysis can be seem below.

4.1 Sentiment Analysis Response Performance Results

Table 13: Sentiment analysis results - multiple models accuracy calculations

Model, Sets	TP / Correct	FP	FN	Recall	Precision	P*R	P+R	F
Facebook								
Cooke	2796	101	103	0.964	0.965	0.931	1.930	0.965
Qian	4581	2726	1603	0.741	0.627	0.464	1.368	0.679
Mody	363759	63555	24388	0.937	0.851	0.798	1.788	0.892
Multiling								
Cooke	2583	352	65	0.975	0.880	0.858	1.856	0.925
Qian	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Mody	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Twitbert								
Cooke	2830	17	153	0.949	0.994	0.943	1.943	0.971
Qian	4871	1527	2512	0.660	0.761	0.502	1.421	0.707
Mody	375347	22096	54259	0.874	0.944	0.825	1.818	0.908

Note: Highest performance results in blue, poorest performance results in red

Table 13 contains the results of testing all 3 candidate models against the 3 candidate datasets. All of the calculations performed are from the equations in S***ection 1.5.6***. The raw values of false positive and negative (FP / FN), recall, precision, and the intermediate calculated values (P*R , P+R) for F score are also represented here. Roberta Hate Speech Dynabench 4, Multilingual Hate Speech Classifier for Social Media Content, and Twitter 2022 154M are all abbreviated to Facebook, Multiling, and Twitbert respectively. Among all models, the best performance was seen from Twitter 2022 154M, followed by the Roberta Hate Speech Dynabench 4, and in last place Multilingual Hate Speech.

For posterity, Multilingual Hate Speech has rows represented with “N/A” for “Not available”. Some of the message contents from the datasets were too large to pass into this model without breaking them apart or altering the model’s defaults. It is disingenuous for performance

analysis to say that the model would work well without tweaking the parameters from their defaults. Thus the model did not produce results for the two larger datasets. Some attempts were made to alter the default inputs to align with the expected configuration on HuggingFace, namely the autotokenizer inputs:

```
tokenizer = AutoTokenizer.from_pretrained(model_name, max_length=512, truncation=True)
```

However, when run against the Qian et al. set, these parameter settings caused the model to split sentiment incorrectly across multiple lines, which made it impossible to perform performance calculations. The Mody et al set was ignored because it likely would produce the same issues as the Qian et al set for this model.

Table 14: Roberta Hate Speech Dynabench 4 Performance Results F-score

Table 14 shows the F-score of the Roberta Hate Speech Dynabench 4 model against all three datasets. Notably performance was highest with the Cooke et al dataset.

Table 15: Twitter 154M Performance Results F-score

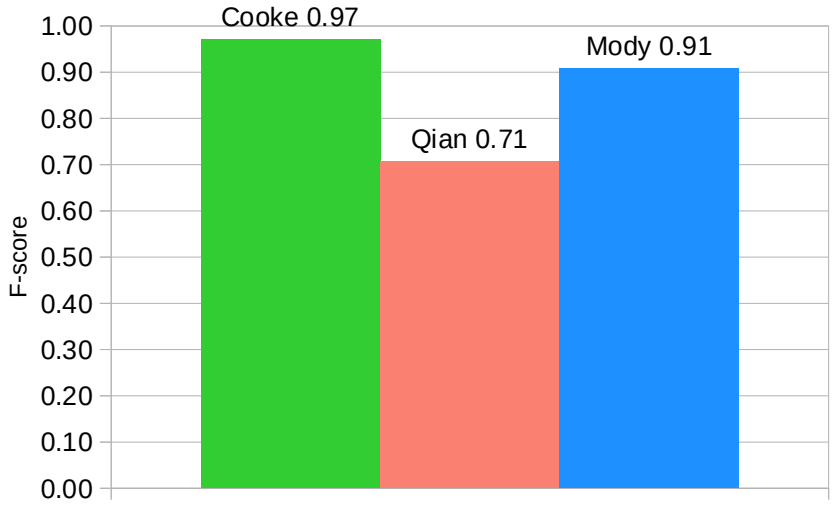

Table 16: Multilingual Hate Speech Performance Results F-score

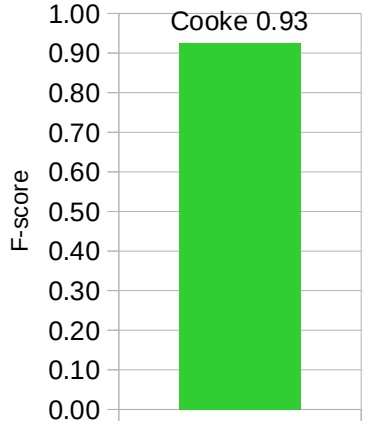

Table 15 shows the F-score of the Twitter 154M model against all three datasets. Notably performance was highest with the Cooke et al dataset as well. ***Table 16*** shows the F-score of the Multilingual Hate Speech model against the set that it was capable of parsing. It yielded a similar performance result to the other two models in this benchmark. Other sets are excluded.

Table 17: Model-Per-Dataset Processing Times and Additional Notes

Model	FB Roberta	Speed	Acc.	Multiling	Speed	Acc.	Twitter Roberta	Speed	Acc
Dataset			% incorrect F-score	Small token window		% incorrect F-score			% incorrect F-score
Cooke 3000 Msg	Large (arbitrary) token maximum	< 1 minute	**%: 6.8 F: 0.9648**	Small max tokens length, can be parsed	~ 3 minutes	**%: 13.90 F: 0.9253**	Large (arbitrary) token maximum	< 1 minute	**%: 5.67 F: 0.9708**
Qian ~9000 msg	“ ”	~ 45 minutes	%: 48.59 F: 0.6791	Produced invalid split sentiments, results can’t be used	~ 210 minutes	N/A	“ ”	~ 1 hour	%: 45.33 F: 0.7069
Mody ~450,000 msg	“ ”	~ 24 hours	%: 19.47 F: 0.8922	N/A	N/A	N/A	“ ”	~ 26 hours	%: 16.91 F: 0.9077

Table 17 contains the amount of time each model needed to process each respective dataset, as well as some additional information about percentage of incorrect sentiment. The best performance scores are represented in bold. Unsurprisingly, larger sets take significantly longer time to process. These tests were run on an 11th Gen Intel i7-11800H with 64 gigabytes of RAM. It is not

entirely certain whether the larger dataset performance analysis was hindered by computer specifications or poor code optimization. Of note is that both Roberta based models took very similar amounts of time to process input. This could be due to several factors including their smaller lexical vocabulary of 50265 versus 250002.

4.2 Analysis of Results

The performance variation of the the two Roberta baseline models from Facebook and the Twitter 154M are the most interesting comparison here. Since they are both trained with a similar base model, this would indicate that only some variation in their training methods has produced a difference in performance when tested against the same datasets. These observed performance differences might be related more to the similarity of test data with the model training data than to the actual ability of models to properly detect content. The Twitter 154M model had a completely different training process than the Roberta Dynabench 4, where it was trained on a corpus of many different pre-labeled sets, with some selection bias for flagging whenever a certain threshold of reviewers labeled something as "hate speech". With Roberta Dynabench 4, their training surprisingly doesn't appear to have much data overlap with the Twitter 154M model, but during final performance review a single dataset was shared between them for performance analysis via Davidson et al., 2017. The Dynabench model performed at just 1% better on the Cooke et al. set (if it can be considered a small benchmark) than its reported final training round score of 95% (Vidgen et al., 2021), which is impressive and reliable accuracy. The Twitter 154M model also produced impressive performance up 8.8% from its reported performance of 87.66% accuracy against the Cooke et al. set as benchmark. Poor performance on the Qian et al. set can be explained by its dataset labeling strategy bias. On the Mody et al. set, it is more difficult to tell why performance is worse, but it could be due to some variation in the data mixed in with such a large set. Potentially the data in that set being less comprised of data from Twitter could be having an impact on the performance as nearly 50% of hate speech data (in larger-scale 63 set analysis) comes from there (Vidgen et al., 2021).

The ranking for all datasets is a clean split on all 3 models for performance with Twitter 154M coming in first, Facebook Dynabench coming in second, and the Multilingual Hate Speech coming in last. The Multilingual Model's paper notes that "multilingual models tend to underperform monolingual models in language-specific tasks" (Barbieri et al., 2022), so this worse difference in performance is expected. However, what is not expected is that the multilingual

model's performance is actually about 0.2 higher in F-score than the paper's prediction range for monolingual analysis of 0.5 – 0.7 for various benchmarks (Barbieri et al., 2022). This model also has seemingly more selection criteria for what constitutes "hate" than the other two models, so this might have an impact on lowering its overall reaction to a given sentiment. Additionally, all of the data in the sets was in English, so this predictably might have had a significant impact on the performance of a model that was not exclusively trained on English data.

4.3 Conclusions of Chapter 4 – Results

The first observation is that the larger dataset sizes took significantly longer to process than the smaller sets. There are a few potential causes for this. Given the computer specs and model complexity, it is not likely that better hardware would produce better results. Rather, code optimization is the most likely reason sentiment is not produced more quickly for large sets. Despite parsing sets in varying batch size, results were roughly the same across sizes and more work would need to be done to properly multithread the parsing for improved performance. Another observation from browsing the message contents of datasets is the actual message string length. The two larger sets had a tendency to have individual messages of many more characters in length. This was particularly noticeable in data from more long-form social media content, such as Reddit posts. This message length increase was perhaps beyond the default vector limit for the multilingual model, so it made performance tests impossible against those larger sets. Interestingly, this is not an actual limitation for that model when used with a bot, due to message length limitations in Telegram.

Both models that were able to process the Qian et al. Reddit and Gab dataset performed very poorly in contrast to the dataset's labeling. After some manual review of some of the individual message contents, it seems clear that the Mechanical Turk process that was used to label this data produced very poor labeling, which made model performance appear much worse for that set.

Also of note is the speed that the Telegram bot was able to respond with. Using a single model, speed seemed to be limited mostly by the performance of the Telegram network itself, rather than the model's ability to produce sentiment quickly for a single message. However, once all 3 models were added, the performance was notably worse. If a bot like this were to be modified to perform parallel processing, this could severely hamper performance. As it is currently written, the bot queues messages and responds to them one at a time.

4.4 Recommendations for Further Research

- Given the performance results from all models, if one were to design a bot for use in actual hate speech content detection activity, it would likely be best to stick with a single highly-performant model or fine-tune a model for a given task. If multiple models are to be used, all models should yield roughly similar performance results to reduce the likelihood of a single model making detection more difficult. It can't be overstressed that models should have their performance benchmarked against labeled data that they were not trained on, in order to better understand their flexibility.
- One of the major things that became apparent after all testing and implementation was the fact that the Telegram bot would not be able to derive sentiment from the *contextual* nature of conversation. This is because the bot can only respond with the sentiment of a single given message. It is beyond the scope of this paper to expand on this, but for further research it could be worthwhile to explore the implementation of a more context-aware bot. Perhaps one that can gather aggregate sentiment over time.
- After the performance analysis, it was already clear that the bootstrap aggregate would not necessarily yield higher accuracy in results at the individual model level. Anecdotally, the difference in model training bias still produces interesting detection results for given single messages. A much larger scale test of the performance of multiple model's bootstrapped scores would be potentially of interest as well.
- Since these tests were all performed using models and data from outside of Telegram, a team with sufficient time and limited bias to adequately label the large quantity of data present in sets like the Pushshift Telegram dataset could produce some very useful training data to use with a bot that is deployed on Telegram, in order to see if model accuracy is just as good with native data.
- Due to poor model performance with sets with specific bias in their labeling, it is important that datasets used in model performance testing have sufficient spread in how they are labeled. It is highly subjective how an individual research team chooses to label their data, and this can have an impact on perceived model performance. It seems that a combination of both manual and machine-learning assisted labeling produces sets with good results, where purely crowd-sourced labeling struggles to produce good data to work with.

5. GENERAL CONCLUSIONS

The major goals of this research were both to test the ability to detect prohibited content on social media platforms, namely hate speech, and also to apply that detection capability in practice. Those goals were achieved and the following conclusions and points have been noted:

1. From the literature analysis and sentiment analysis bot design consideration sections it is clear that Social Media platforms (particularly Telegram) have a great need for content detection as a method for platform governance assistance. The analysis showed that many different strategies exist for content detection. Not only that, but the ways in which they are applied are also numerous. Dissecting current research not only helped to uncover tools, rationale, and methods for both detecting natural language content sentiment. It also helped practically for applying this sentiment through a Telegram bot.

2. Based on the research, a dual-approach design for a bot that can perform hate speech sentiment analysis is proposed. This design is driven both by the practical needs of implementation stemming from the unique characteristics of machine learning assisted sentiment analysis and also the quirks of this application in the context of a bot. The proposal workflows help to contextualize and decompose the different steps required for this implementation and how results can be adequately determined from the two proposals. These strategies fundamentally drove final implementation and results collection.

3. After datasets were selected during discovery in literature analysis phase, they could be analyzed and processed in a way that would be suitable for sentiment analysis with trained machine learning models. The sentiment analysis performance was tested in this phase alongside the practical implementation of the code that would drive the bot utilizing this sentiment analysis. Results show the performance characteristics of the models on data which they were not trained and also the application of that performance in context.

4. The performance results indicate that not all models, though trained on similar data, are capable of high performance when used against data on which they were not trained. They also show some difference of inherent bias that would otherwise not be clear without external performance tests. Clearly there is an impact associated with the methodologies and parameters used for model training that affects their ability to perform outside of initial training context. This includes a more practical application like a content detection bot.

This thesis highlights some of the significant issues that modern social networks face in the context of platform governance. The analysis also showed that many proprietary methods may not align with academic understanding of the effectiveness of content detection strategy in isolation.

Content detection methods are not perfect and the results found from this research serve to show that often those methods, without the context of practical application, have mixed variance in their ability to perform as expected. The purpose of this thesis was to make it clear that a practical implementation of these methods, when found to be performant, is both feasible and potentially highly extensible. Recommendations have been made for further analysis of the performance of the growing field in the use of these detection methods. Additionally, some strategies for increasing the effectiveness of the implementation of these methods with vectors such as bots have been proposed. This is all to promote a clearer and open academic understanding of these tools or the “algorithm” that frequently drives proprietary content detection strategies.

REFERENCES

Antypas, D., & Camacho-Collados, J. (2023). Robust Hate Speech Detection in Social Media: A Cross-Dataset Empirical Evaluation. In Y. Chung, P. R\textbackslash"ottger, D. Nozza, Z. Talat, & A. Mostafazadeh Davani (Eds.), *The 7th Workshop on Online Abuse and Harms (WOAH)* (pp. 231–242). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.woah-1.25

Badiei, F. (2020). The Tale of Telegram Governance: When the Rule of Thumb Fails. *The Justice Collaboratory*, 33.

Barbieri, F., Anke, L. E., & Camacho-Collados, J. (2022). *XLM-T: Multilingual Language Models in Twitter for Sentiment Analysis and Beyond* (arXiv:2104.12250). arXiv. https://doi.org/10.48550/arXiv.2104.12250

Baumgartner, J., Zannettou, S., Squire, M., & Blackburn, J. (2020). The Pushshift Telegram Dataset. *Proceedings of the International AAAI Conference on Web and Social Media*, *14*, 840–847. https://doi.org/10.1609/icwsm.v14i1.7348

Bickert, M. (2022, August 25). Meta Community Standards Enforcement Report, Second Quarter 2022. *Meta*. https://about.fb.com/news/2022/08/community-standards-enforcement-report-q2-2022/

Bots: An introduction for developers. (n.d.). Retrieved December 20, 2022, from https://core.telegram.org/bots

Bovet, A., & Grindrod, P. (2022). Organization and evolution of the UK far-right network on Telegram. *Applied Network Science*, *7*(1), Article 1. https://doi.org/10.1007/s41109-022-00513-8

CAS > Home. (n.d.). Retrieved December 20, 2022, from https://cas.chat/

ChatGPT. (2022, November 30). *Https://Openai.Com/Blog/Chatgpt/*. https://openai.com/blog/chatgpt/

Combot. (n.d.). Retrieved December 20, 2022, from https://combot.org/

Cooke, S., Graux, D., & Dev, S. (2023). Multi Platform-Based Hate Speech Detection: *Proceedings of the 15th International Conference on Agents and Artificial Intelligence*, 541–549. https://doi.org/10.5220/0011698600003393

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). *Automated Hate Speech Detection and the Problem of Offensive Language* (arXiv:1703.04009). arXiv. https://doi.org/10.48550/arXiv.1703.04009

Decompression APIs—Python-zstandard 0.22.0-pre documentation. (n.d.). Retrieved June 17, 2023, from https://python-zstandard.readthedocs.io/en/latest/decompressor.html

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. https://doi.org/10.48550/arXiv.1810.04805

Gaikwad, D. P., & Thool, R. C. (2015). Intrusion Detection System Using Bagging Ensemble Method of Machine Learning. *2015 International Conference on Computing Communication Control and Automation*, 291–295. https://doi.org/10.1109/ICCUBEA.2015.61

GIFCT. (2019). https://perma.cc/44V5-554U

Godoy, D. (2022, July 10). *Understanding binary cross-entropy / log loss: A visual explanation*. Medium. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Gomez, R., Gibert, J., Gomez, L., & Karatzas, D. (2019). *Exploring Hate Speech Detection in Multimodal Publications* (arXiv:1910.03814). arXiv. http://arxiv.org/abs/1910.03814

Gorwa, R., Binns, R., & Katzenbach, C. (2020). Algorithmic content moderation: Technical and political challenges in the automation of platform governance. *Big Data & Society*, *7*(1), 2053951719897945. https://doi.org/10.1177/2053951719897945

Hate Speech | Meta Transparency Center. (2024, January 12). Facebook (Meta) Community Standards. https://web.archive.org/web/https://transparency.fb.com/policies/community-standards/hate-speech/

Jarynowski, A., Semenov, A., Kamiński, M., & Belik, V. (2021). Mild Adverse Events of Sputnik V Vaccine in Russia: Social Media Content Analysis of Telegram via Deep Learning. *Journal of Medical Internet Research*, *23*(11), e30529. https://doi.org/10.2196/30529

Jégou, H., Perronnin, F., Douze, M., Sánchez, J., Pérez, P., & Schmid, C. (2012). Aggregating Local Image Descriptors into Compact Codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(9), 1704–1716. https://doi.org/10.1109/TPAMI.2011.235

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260. https://doi.org/10.1126/science.aaa8415

Justus, D., Brennan, J., Bonner, S., & McGough, A. S. (2018). Predicting the Computational Cost of Deep Learning Models. *2018 IEEE International Conference on Big Data (Big Data)*, 3873–3882. https://doi.org/10.1109/BigData.2018.8622396

Kahn, R., & Das, A. (2018). *Build Better Chatbots: A Complete Guide to Getting Started with Chatbots*. Apress.

Koro, D. (2019). *Botanicum* [Python]. https://github.com/vienmoir/Botanicum (Original work published 2019)

Korotaeva, D., Khlopotov, M., Makarenko, A., Chikshova, E., Startseva, N., & Chemysheva, A. (2018). Botanicum: A Telegram Bot for Tree Classification. *2018 22nd Conference of Open Innovations Association (FRUCT)*, 88–93. https://doi.org/10.23919/FRUCT.2018.8468278

Language Understanding (LUIS) | Microsoft Azure. (n.d.). Retrieved January 21, 2023, from https://azure.microsoft.com/en-us/products/cognitive-services/conversational-language-understanding/

Larsen, P. (2022). *Tgbot* [Python]. https://github.com/PaulSonOfLars/tgbot (Original work published 2017)

Llansó, E. J. (2020). No amount of "AI" in content moderation will solve filtering's prior-restraint problem. *Big Data & Society*, *7*(1), 2053951720920686. https://doi.org/10.1177/2053951720920686

lorien. (2022). *Lorien/daysandbox_bot* [Python]. https://github.com/lorien/daysandbox_bot (Original work published 2017)

MacAvaney, S., Yao, H.-R., Yang, E., Russell, K., Goharian, N., & Frieder, O. (2019). Hate speech detection: Challenges and solutions. *PLOS ONE*, *14*(8), e0221152. https://doi.org/10.1371/journal.pone.0221152

Machine Learning – Amazon Web Services. (n.d.). Amazon Web Services, Inc. Retrieved June 4, 2023, from https://aws.amazon.com/sagemaker/

Miss Rose. (n.d.). Retrieved November 4, 2022, from https://missrose.org/

MMDetection Contributors. (2018). *OpenMMLab Detection Toolbox and Benchmark* [Python]. https://github.com/open-mmlab/mmdetection (Original work published 2018)

Modrzyk, N. (2019). *Building Telegram Bots: Develop Bots in 12 Programming Languages using the Telegram Bot API*. Apress. https://doi.org/10.1007/978-1-4842-4197-4

Mody, D., Huang, Y., & Alves de Oliveira, T. E. (2023). A curated dataset for hate speech detection on social media text. *Data in Brief*, *46*, 108832. https://doi.org/10.1016/j.dib.2022.108832

Multimedia Commons—Registry of Open Data on AWS. (n.d.). Retrieved January 21, 2023, from https://registry.opendata.aws/multimedia-commons/

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. *Journal of the American Medical Informatics Association*, *18*(5), 544–551. https://doi.org/10.1136/amiajnl-2011-000464

NLTK :: Natural Language Toolkit. (n.d.). Retrieved January 21, 2023, from https://www.nltk.org/

OpenAI, Blog, GPT3-Apps. (2021, March 25). GPT-3 Powers the Next Generation of Apps. https://openai.com/blog/gpt-3-apps/

Packeer, S., & Kannangara, D. T. V. (2022). Detection of pedophilia content online: A case study using Telegram. *Iraqi Journal For Computer Science and Mathematics*, *3*(2), Article 2. https://doi.org/10.52866/ijcsm.2022.02.01.007

Pak, A., & Paroubek, P. (2010, May). Twitter as a Corpus for Sentiment Analysis and Opinion Mining. *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. LREC 2010, Valletta, Malta. http://www.lrec-conf.org/proceedings/lrec2010/pdf/385_Paper.pdf

Patil, A., Marimuthu, Nagaraja Rao, & Niranchana. (2017). Comparative study of cloud platforms to develop a Chatbot. *International Journal of Engineering & Technology*, *6*(3), 57. https://doi.org/10.14419/ijet.v6i3.7628

Peeters, S., & Willaert, T. (2022). Telegram and Digital Methods: Mapping Networked Conspiracy Theories through Platform Affordances. *M/C Journal*, *25*(1), Article 1. https://doi.org/10.5204/mcj.2878

Python-telegram-bot. (n.d.). Retrieved November 25, 2022, from https://python-telegram-bot.org/

PyTorch. (n.d.). Retrieved January 21, 2023, from https://www.pytorch.org

Qian, J., Bethke, A., Liu, Y., Belding, E., & Wang, W. Y. (2019). *A Benchmark Dataset for Learning to Intervene in Online Hate Speech* (arXiv:1909.04251). arXiv. https://doi.org/10.48550/arXiv.1909.04251

Raj, S. (2019). *Building Chatbots with Python: Using Natural Language Processing and Machine Learning*. Apress. https://doi.org/10.1007/978-1-4842-4096-0

Serverless Computing – AWS Lambda Pricing – Amazon Web Services. (n.d.). Amazon Web Services, Inc. Retrieved June 5, 2023, from https://aws.amazon.com/lambda/pricing/

Shah, D., Harrison, T. G., Freas, C. B., Maimon, D., & Harrison, R. W. (2020). Illicit Activity Detection in Large-Scale Dark and Opaque Web Social Networks. *2020 IEEE International Conference on Big Data (Big Data)*, 4341–4350. https://doi.org/10.1109/BigData50022.2020.9378229

Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. In A. Sattar & B. Kang (Eds.), *AI 2006: Advances in Artificial Intelligence* (pp. 1015–1021). Springer. https://doi.org/10.1007/11941439_114

spaCy. (n.d.). Retrieved January 21, 2023, from https://spacy.io/

Taboada, M. (2016). Sentiment Analysis: An Overview from Linguistics. *Annual Review of Linguistics*, *2*(1), 325–347. https://doi.org/10.1146/annurev-linguistics-011415-040518

Taha, A. A., & Hanbury, A. (2015). Metrics for evaluating 3D medical image segmentation: Analysis, selection, and tool. *BMC Medical Imaging*, *15*, 29. https://doi.org/10.1186/s12880-015-0068-x

Telegram APIs. (n.d.). Retrieved November 25, 2022, from https://core.telegram.org/api

Telegram Bot Daysandbox Bot. (n.d.). TgDev. Retrieved December 20, 2022, from https://tgdev.io/bot/daysandbox_bot

Telegram FAQ. (n.d.). Telegram. Retrieved November 4, 2022, from https://telegram.org/faq#q-what-makes-telegram-groups-cool

Telegram Press Info. (n.d.). Telegram. Retrieved November 4, 2022, from https://telegram.org/press

TensorFlow. (n.d.). TensorFlow. Retrieved January 18, 2023, from https://www.tensorflow.org/

Terms of Service. (n.d.). Telegram. Retrieved November 27, 2022, from https://telegram.org/tos

Transformers—State-of-the-art Machine Learning for PyTorch, TensorFlow, and JAX. (n.d.). [Documentation]. Hugging Face. Retrieved December 20, 2022, from https://huggingface.co/docs/transformers/index

Twitter API Documentation. (n.d.). Retrieved December 20, 2022, from https://developer.twitter.com/en/docs/twitter-api

Uc-Cetina, V., Navarro-Guerrero, N., Martin-Gonzalez, A., Weber, C., & Wermter, S. (2022). Survey on reinforcement learning for language processing. *Artificial Intelligence Review*. https://doi.org/10.1007/s10462-022-10205-5

Vidgen, B., Botelho, A., Broniatowski, D., Guest, E., Hall, M., Margetts, H., Tromble, R., Waseem, Z., & Hale, S. (2020). *Detecting East Asian Prejudice on Social Media* [dataset]. Zenodo. https://doi.org/10.5281/zenodo.3816667

Vidgen, B., Thrush, T., Waseem, Z., & Kiela, D. (2021). *Learning from the Worst: Dynamically Generated Datasets to Improve Online Hate Detection* (arXiv:2012.15761). arXiv. https://doi.org/10.48550/arXiv.2012.15761

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, 1096–1103. https://doi.org/10.1145/1390156.1390294

Wang, B., Zhang, L., Wen, L., Liu, X., & Wu, Y. (2021). Towards Real-World Prohibited Item Detection: A Large-Scale X-ray Benchmark. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 5392–5401. https://doi.org/10.1109/ICCV48922.2021.00536

Wong, D. (2021). *Real-World Cryptography*. Manning.

Word2vec | TensorFlow Core. (n.d.). TensorFlow. Retrieved January 21, 2023, from https://www.tensorflow.org/tutorials/text/word2vec