END-TO-END MESSAGING SYSTEM ENHANCEMENT USING FEDERATED LEARNING FOR CYBERBULLYING DETECTION

A PROJECT REPORT

submitted by

AJAY S RAM (TCR18CS003)
AMARNATH C N (TCR18CS010)
KOWSIK NANDAGOPAN D (TCR18CS031)
NAVANEETH D (TCR18CS043)

to

the APJ Abdul Kalam Technological University
in partial fulfilment of the requirements for the award of the Degree

of

Bachelor of Technology
in
Computer Science and Engineering

Department of Computer Science and Engineering

Government Engineering College, Thrissur
Thrissur – 680009

JUNE, 2022

DECLARATION

I, on behalf of authors of the report: Ajay S Ram (TCR18CS003), Amarnath C N (TCR18CS010), Kowsik Nandagopan D (TCR18CS031), Navaneeth D (TCR18CS043), hereby declare that the project report "End-to-End Messaging System enhancement using Federated Learning for Cyberbullying Detection" submitted for partial fulfilment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University , Kerala is a bonafide work done by us under supervision of Ms. Bisna N D, Assistant Professor. This submission represents our ideas in our own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also invoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Thrissur
Date: 19-05-2022

Ajay S Ram (TCR18CS003)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GOVERNMENT ENGINEERING COLLEGE, THRISSUR
Thrissur – 680009

CERTIFICATE

This is to certify that the report entitled **'End-to-End Messaging System enhancement using Federated Learning for Cyberbullying Detection'** submitted by **Ajay S Ram (TCR18CS003), Amarnath C N (TCR18CS010), Kowsik Nandagopan D (TCR18CS031), Navaneeth D (TCR18CS043)** to the APJ Abdul Kalam Technological University in partial fulfilment of the requirements for the award of the Degree of Bachelor Technology in Computer Science and Engineering is a bonafide record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Internal Supervisor(s)

Department Seal

Project Coordinator(s) Head of the Department

ACKNOWLEDGEMENTS

We express our deep sense of gratitude to our respected and learned guide, Ms. Bisna N. D, Assistant Professor, Department of Computer Science and Engineering. and our project coordinator Mr. George Mathew, Assistant Professor, Department of Computer Science and Engineering. for their valuable help and guidance. We are thankful to them for the encouragement they have given us in completing this project. We are also grateful to respected Ms. Shibily Joseph, Head of Depatment, Department of Computer Science and Engineering for permitting us to utilise all the necessary facilities of the department.

We are also thankful to all the other faculty and staff members of our department for their kind co-operation and help.

We also thank our friends and other members of the college for their advice, support and constructive criticism.

ABSTRACT

Social media has shortened the digital world significantly. Though it has facilitated remote communication, the implicit dangers of such an application is not something to be overlooked. Spread of hate-speech and offensive text through social media have catalysed depression and suicidal tendencies especially among teenagers. In light of these concerns, we propose *Schat*, an End-to-End messaging system with inbuilt measures to counter hate speech/cyberbullying while protecting the privacy of the user.

The project uses Natural Language Processing (NLP) to correctly classify and warn users about offensive text usage in our End-to-End platform. Since the level of offensiveness is subjective, conventional sentiment analysis might not do a perfect job in classifying them. A way to get around this is to use significantly large and diverse Deep Learning datasets that can generalize the model. But with the increasing sensitivity of data ownership and large processing power and time needed for training, such methods have become less popular in the AI industry. In this scenario, Federated learning has a huge potential for reconciling the need for enormous Deep Learning datasets. This privacy-preserving approach is a distributed machine learning technique in which client devices train models locally without sharing any data, except for parameter changes, which get aggregated to a central model. Our work has materialised such a Federated Learning system which showcases a very good training and testing performance on real-world texts containing offensive words.

While using our application, a person encountering an offensive text has the provision to tag it, if our model predicts the offensiveness to be greater than a certain threshold. Then the model is trained on the tagged text from the client side and the parameter updates are sent to the central model ensuring the confidentiality of our user's data.

CONTENTS

Contents | Page No.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 CURRENT SYSTEM

Almost all social media are looking into methods to reduce cyber harassment and spreading of false information, the approach is almost similar in all platforms. First the data has to be collected from the users and some machine learning models classify it into separate categories. If the content is found to have a negative impact it is alerted to all the remaining users. Whatsapp uses an identified count to prevent spreading of fake messages, twitter does sentiment analysis on the users tweets etc. The drawback of such system is that; first, large computing structure needs to be setup and the result generated from it is for a general public rather than a specific user; next the incident response for such messages take time from both law-enforcement officials as well as the social media platform

1.1.1 Limitations of existing solutions

1. Increased incident response time
2. Bulk data collection from user
3. Privacy concern
4. Takes time to train the machine learning model

1.2 PROPOSED SYSTEM

The project uses Natural Language Processing (NLP) to correctly classify and warn users about offensive text usage in our end-to-end platform. Since the level of offensiveness is subjective, conventional sentiment analysis might not do a perfect job in classifying them. A way to get around this is to use significantly large and diverse Deep Learning datasets that can generalize the model. But with the increasing sensitivity of data ownership and large processing power and time needed for training, such methods have become less popular in the AI industry. In this scenario, Federated learning has a huge potential for reconciling the need for enormous Deep Learning datasets. This privacy-preserving approach is a distributed machine learning technique in which client devices train models locally without sharing any data, except for parameter changes, which get aggregated to a central model. While using our application, a person encountering an offensive text has the provision to tag it, if our model predicts the offensiveness to be greater than a certain threshold. Then the model is trained on the tagged text from the client side and the parameter updates are sent to the central model ensuring the confidentiality of our user's data.

1.3 FEASIBILITY

1.3.1 Technical Feasibility

We have analyzed the technical feasibility of the project based on the following factors.

(i) Software Feasibility

- Operating System - Ubuntu 20.04 LTS
- Federated Machine learning - Flower Framework, Pytorch
- Authentication Server - Python Django

- Federated Aggregation Server - Python Django
- Client Backend - Python Django
- Client User Interface - React JS (Javascript)
- Collaboration - GitHub, Git
- Project Management - GitHub Organization, Project Libre

Details of software stack is provided in Section 3.1.2

(ii) Hardware Feasibility

No hardware is being used in this project directly.

1.3.2 Economical & Financial Feasibility

Initial cost for production release is provided in Table 1.1 [1]. After initial deployment we have to scale the server horizontally. Monthly running cost will be nearly 20,000 INR. In our project, we have used a free tier of services mentioned in the table. We have planned to develop using Open Source software hence we have reduced project cost significantly.

Price is calculated using pricing calculator provided in official websites,

- **Colab Pro +** `https://colab.research.google.com/signup`
- **Azure Pricing Calculator** `https://azure.microsoft.com/en-us/pricing/calculator/`

[1]Cost is estimated for 4 months (estimated project duration - including development, testing and demonstration). As per variations in USD - INR rate, project cost may change.

1.3.3 Schedule Feasibility

The final year project schedules for the seventh and eighth semesters are shown in the Figure 1.1 and Figure 1.2 respectively. Green check marks in Gantt chart indicates items that have been completed.

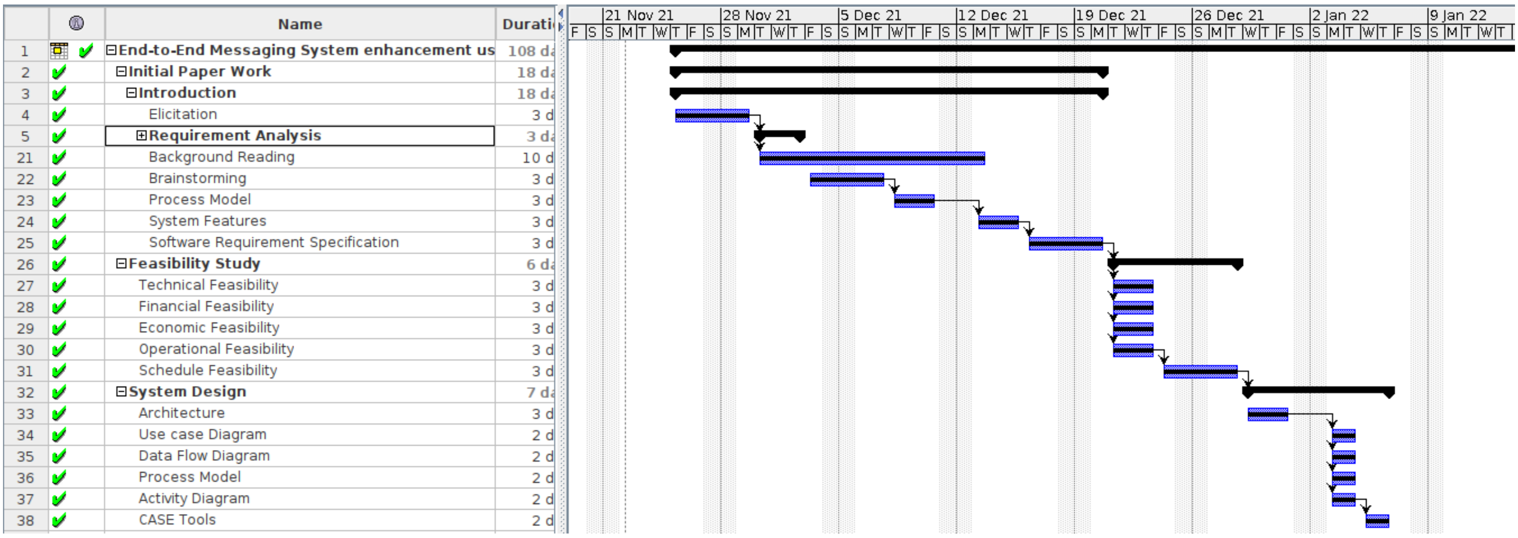

Figure 1.1: Phase 1 - Gantt Chart

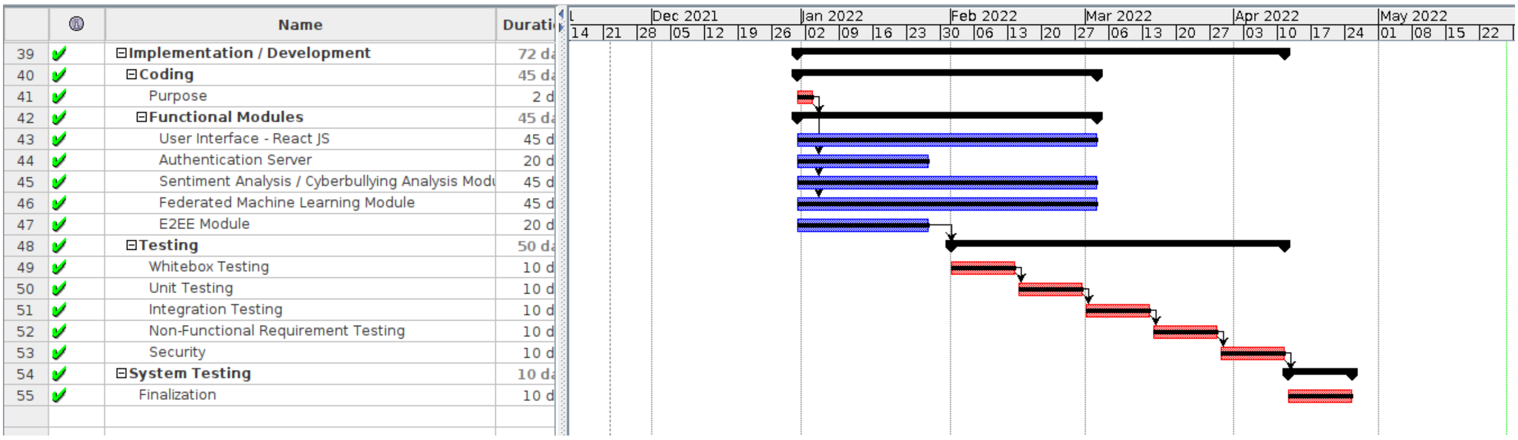

Figure 1.2: Phase 2 - Gantt Chart

1.3.4 Resource Feasibility

Authentication server can run on 1 vCPU, 1GB RAM smoothly on any cloud infrastructure. For our project we have opted Oracle Cloud. For later phases, the deployment would be migrated to Microsoft Azure. Azure provides on-demand computing services at a reasonable cost. By enabling auto-scaling features and close monitoring on the usage spikes, it is possible to scale the same application with little changes. On the client-side, the minimum requirement is a laptop or PC, with at least 500MB of free space, 4GB RAM, a CPU and decent internet connectivity even after scaling up the application.

1.4 PROCESS MODEL

Process model abstracts the software development process by specifying the various stages involved along with their sequence. It gives us a glimpse on the various tasks that should be undertaken and their corresponding inputs and outputs. Choosing a process model is affected by several factors like the project requirement, size, complexity, cost of delay, customer involvement, familiarity with technology and project resources. Although the project requires constant monitoring and feedback from the clients especially regarding the robustness of our Federated learning model, the overall requirement of the project seldom needs change. The project is not highly time-bound. Additionally, client involvement is critical in various rounds of the federated learning process. The project requires a meager amount of resources, like funds and staff, during its initial stages. Although scaling may widely change the resources needed. Since our project involves one of the most widely researched areas in AI, a Spiral model would be the best alternative as it is still in Research and development. The spiral model is a risk driven iterative software process model. The spiral model delivers projects in loops. Unlike other process models, its steps aren't activities but phases for addressing whatever problem has the greatest risk of causing a failure. This model is apt for managing uncertainty as it involves analyzing the highly-risked problem, evaluating the alternatives and developing the solution before planning for the next cycle.

Table 1.1: Deployment Cost

Product	Use	Cost per month (Including GST)	Cost for project completion
Colab Pro + (With GPU)	Initial machine learning model training.	60 USD (4508.95 INR)	60 USD (18035.78 INR)
Azure Virtual Machine (Without GPU)	Aggregation server, Notification server, Authentication server	80 USD (6011.93 INR)	80 x 4 months = 320 USD (18035.78 INR)
Azure Student Plan Discount			- 100 USD (- 7514.91 INR)
Total cost			**280 USD (21041.75 INR)**

CHAPTER 2
REQUIREMENT ANALYSIS

2.1 INTRODUCTION

2.1.1 Document Purpose

Software Requirement Specifications (SRS) in this document are for the End-to-End Messaging System enhancement using Federated Learning for Hate Cyberbullying detection. The important benefit of doing this project is that we can control cyberbullying and hate speech from spreading in social media without losing any privacy. Purpose of the document is to study the requirements and methods that will effectively and efficiently meet the purpose.

2.1.2 Product Purpose

Purpose of our project is to reduce hate speech from spreading in social media, especially in end-to-end systems where either there is no server or the server stores encrypted texts. Our methods will guarantee privacy by keeping the messages in the user's device while still able to train the model effectively. This project also aims to help law enforcement agencies by reducing the workload on their side.

2.1.3 Intended Audience and Document Overview

Our project is intended to those who have felt the need to eliminate hate speech from social communication.

2.1.4 Identifying the Stakeholders

Stakeholders are those people who are affected by a project or result. It is the responsibility of the project deployment and implementation team to determine stakeholders, their requirements and expectations. Stakeholders that we have identified for this project are:

- **Development Team** - People who are responsible for managing risk, planning, testing, analyzing, programming and other activities throughout the course of the project
- **Users** - Common social media users
- **Law Enforcement Agencies** - as the result of our project reduces the workload on these agencies.

2.1.5 Setting the goals

The questions given below will help the development team to identify the problem and will help in coming up with an appropriate solution.

- What problem does this project solve?
- What does the final solution look like?
- What are the actual benefits gained by adopting this solution?
- What are the risk factors that we should take into consideration?
- Cost estimation
- Hardware and software availability

- Technical skills required
- Acceptance of the product
- How should the problem be solved?

2.2 METHOD OF REQUIREMENT ELICITATION

The techniques we used for gathering requirements from the users are given below.

- Questionnaire through Google Forms
- Poll conducted through Instagram
- Study of existing techniques and documents

2.2.1 Questionnaire Through Google Forms

We delivered many questions related to hate speech through google forms and circulated to many people and collected detailed reports on how people see hate speeches and how it affects the people. Figure 2.1 to Figure 2.3 shows results after analyzing the survey.

2.2.2 Poll Conducted Through Instagram Story

Apart from the survey through google forms,we conducted an instagram poll to know more about the cyberbullying concepts of the common people. Figure 2.4 shows the results of the poll.

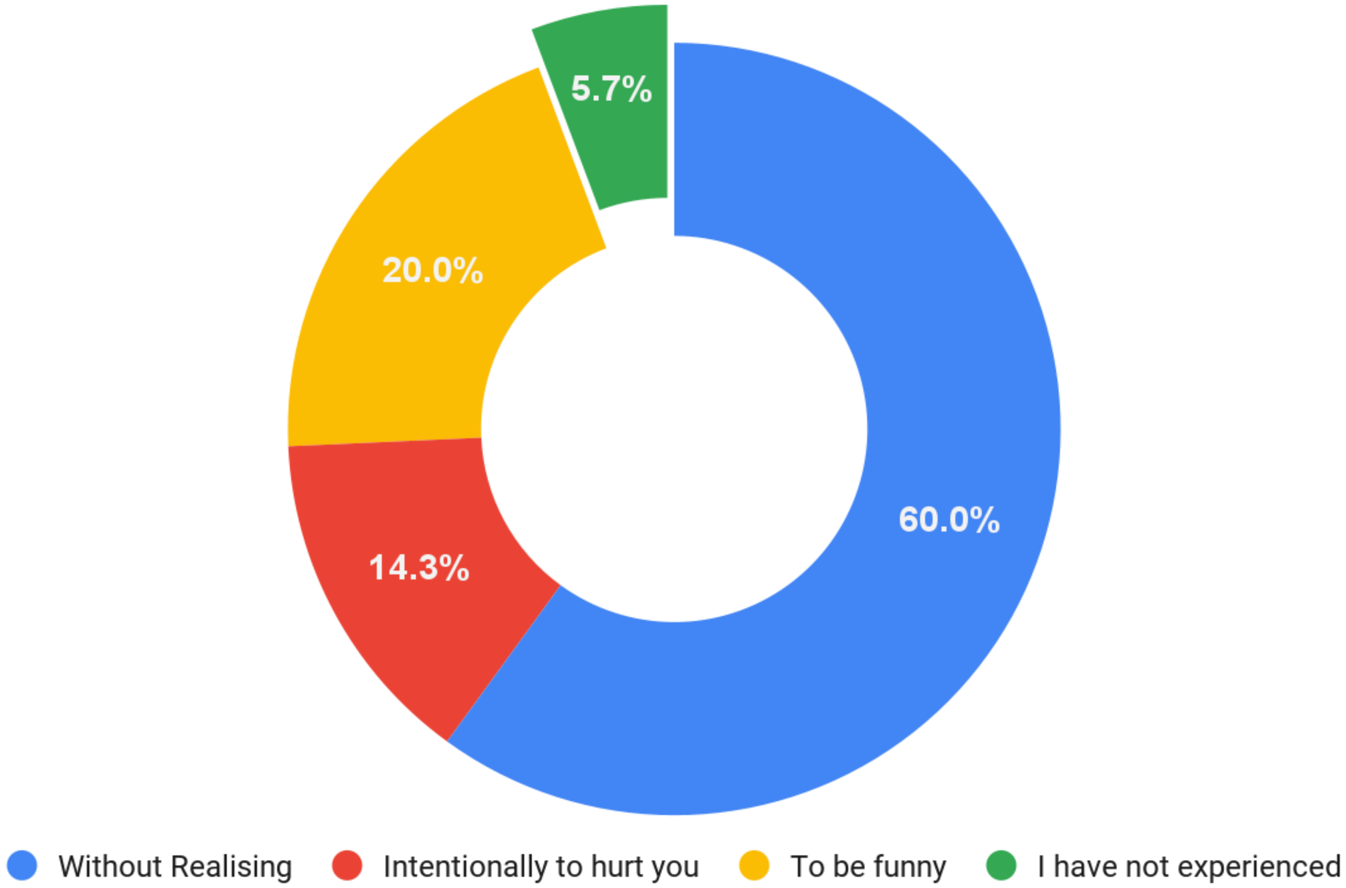

Figure 2.1: 94.1 percent of the people we have surveyed experienced cyberbullying in one way or the other

2.3 USER REQUIREMENTS

On the basis of requirement analysis conducted through Google Form polls, Instagram polls and detailed study on implemented system, it is clear that most of the users are not satisfied with the current system to prevent cyberbullying. The amount of time taken for the result is long and it's too difficult for the current system to prevent the spreading of hatred.

Currently users need both these features. That is a hate speech should be

- Detected as soon as possible

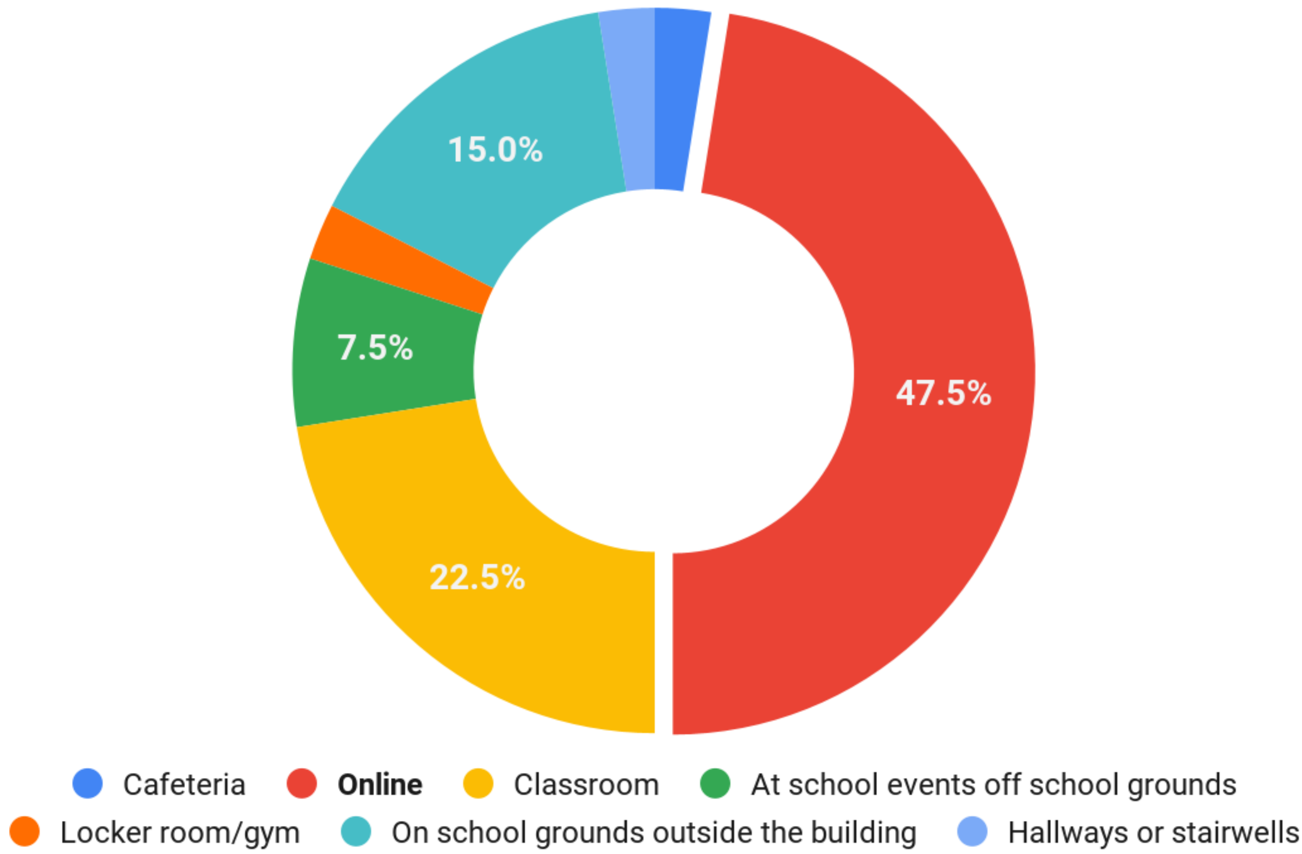

Figure 2.2: 47.5 percent of the people we have surveyed were cyberbullied Online

- Removed as soon as possible

- Should ensure security

2.4 PROJECT REQUIREMENTS

On the basis of requirements demanded by the user, the following project requirements are found out.

- Functionality to detect hate speeches and cyberbullying.

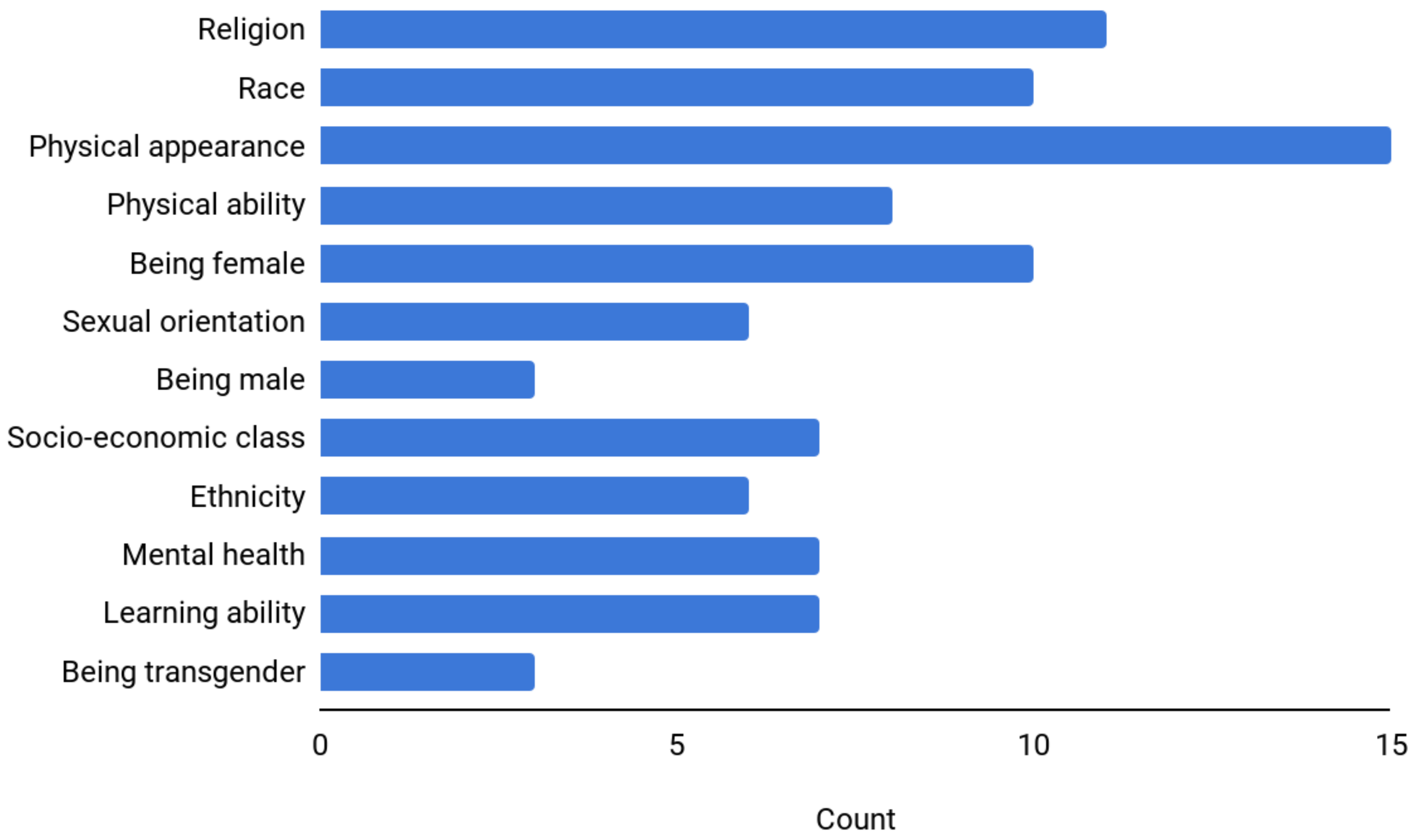

Figure 2.3: Most of the people we surveyed experienced cyberbullying due to their physical appearance

- Functionality to automatically delete these types of messages and thereby preventing the spread of messages as soon as possible.
- Privacy of every user should be respected and preserved.

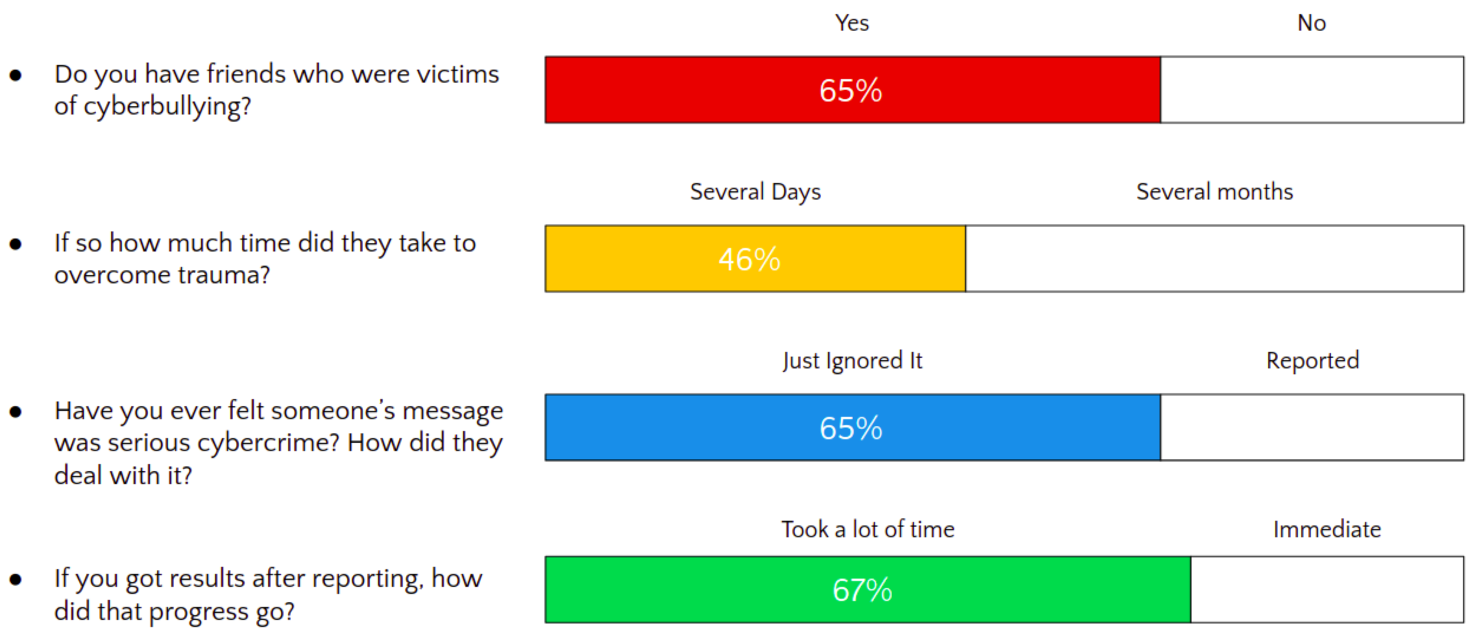

Figure 2.4: Questions and polls from instagram survey

CHAPTER 3
DESIGN AND IMPLEMENTATION

3.1 ARCHITECTURAL DESCRIPTION

3.1.1 Introduction

Figure 3.1 abstracts our project architecture. Here the Federated Learning algorithms and application interface constitutes the Software interfaces while the client device and the central server make up the hardware components

- **Software Interfaces** - The software interface of the system consists mainly of an Transformer-based Sentiment Analysis model that effectively identifies and classifies Hate-Speech in a given text. Additionally, we use Federated Learning algorithms and aggregators to collectively train the model across a number of client devices and aggregate their local updates so as to optimize the global model.
- **Hardware Interfaces** - Since we are dealing with cross-device network type as compared to cross-silo architecture, the hardware components mainly consist of many client devices with limited computational capability.

(i) **Decomposition Description**

The System is divided into 5 modules based on the functionality of the system. A concise representation of the modules is illustrated in Figure 3.2. The Modules are :

- **Authentication server** - Identity of users are to be validated before actual working.Authentication server deals with this process.The server verifies user's identity

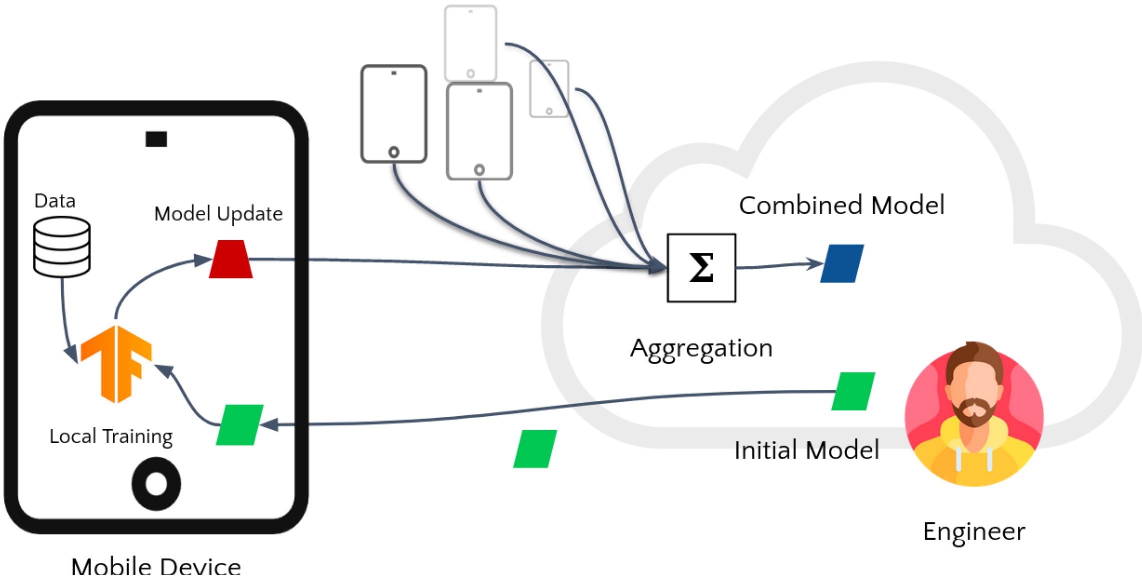

Figure 3.1: Federated Learning Architecture

by crosschecking with credentials.

- **Sender side** - Sender side deals with sending of messages adhering to end-to-end policy. Messages are first encrypted with the sender's private key then encrypted with receiver's public key.

- **Receiver side** - Receives incoming messages after decryption we pass the message through the model to check whether the messages are offensive or not. We will first decrypt using receiver's private key then decrypt with sender's public key. This ensures confidentiality and authenticity of the message.

- **Aggregation server** - The Aggregation server is responsible for aggregating all the local updates coming from individual clients to one global update. This update is then passed back to individual devices to keep the local model synchronised with the global model.

- **Notification server** - Updates should be notified to all the users in the group for proper working of the whole idea. These updates are send to the users by notification server in a fixed period of time.

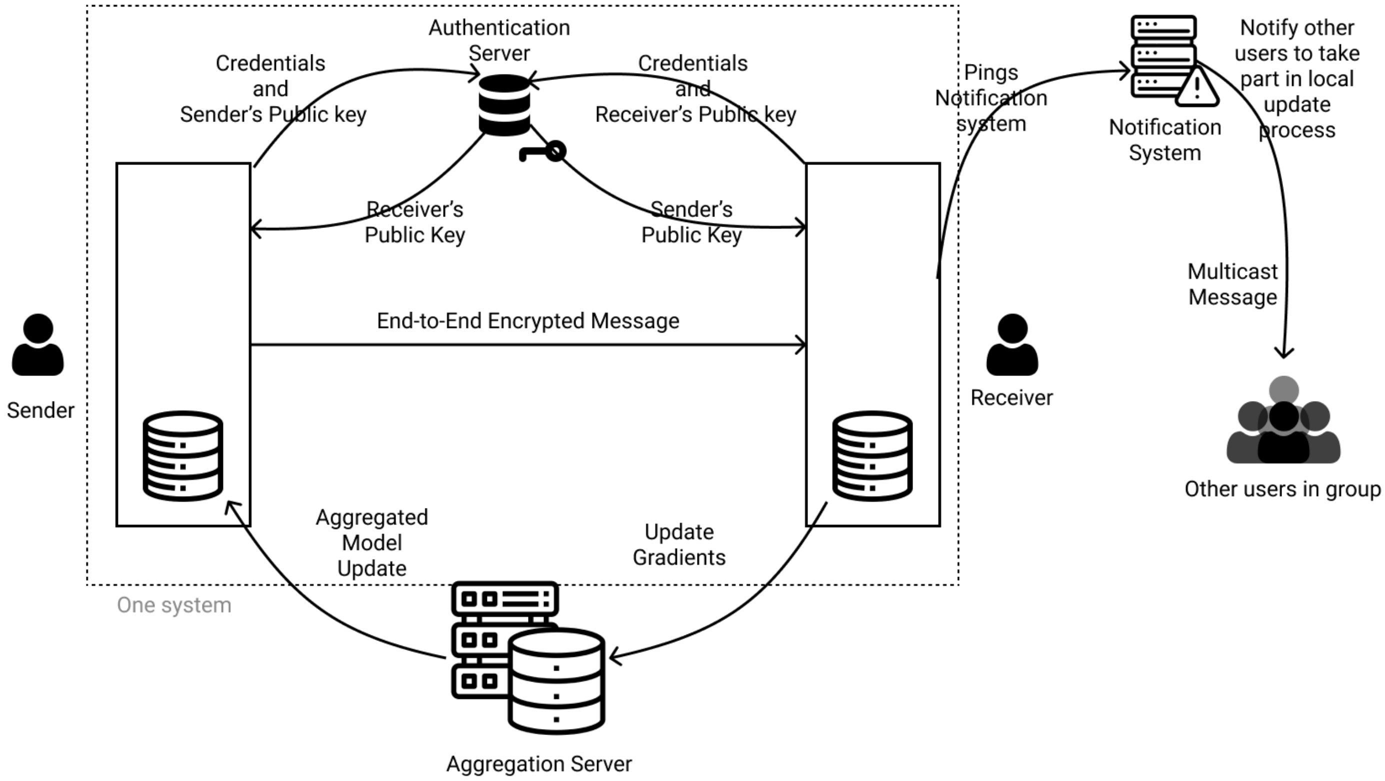

Figure 3.2: Modularized Architecture

3.1.2 Software Stack Architecture

Software stack is shown in Figure 3.3. It is quite complex architecture. Considering the maintainability we have reused and tried to minimize the technology as much as possible. One client consists of a user interface, and a backend. User interface is designed using React JS. Backend of the user side is responsible for avoiding computation overhead and classification of messages. Flower framework is used to design Federated machine learning parts and Pytorch for building machine learning models. User interface interacts with the backend using JSON APIs provided by Django server connecting flower framework.

Public key distribution server is built using python Django. SQLite database is used to store keys, user profile, user login credentials etc.

Model aggregation server is also built using Django in combination with flower framework. This server aggregates individual models sent by the clients and generates a single model using FedAvg strategy.

Users can sent messages from the React User Interface to another user using Firebase realtime database service.

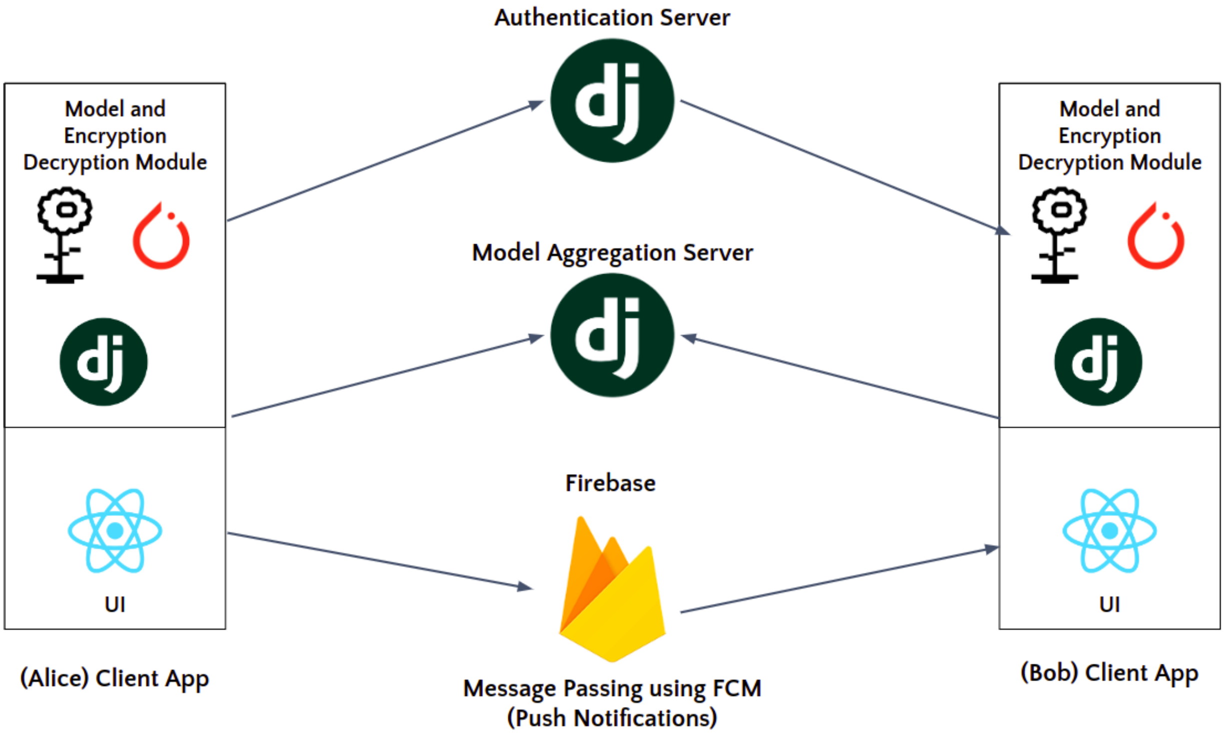

Figure 3.3: Software Stack Architecture

3.1.3 Use case diagram

A use case diagram is a graphic depiction of the interaction among the elements of a system. A use case is a methodology used in system analysis to identify, clarify and organize system requirements.

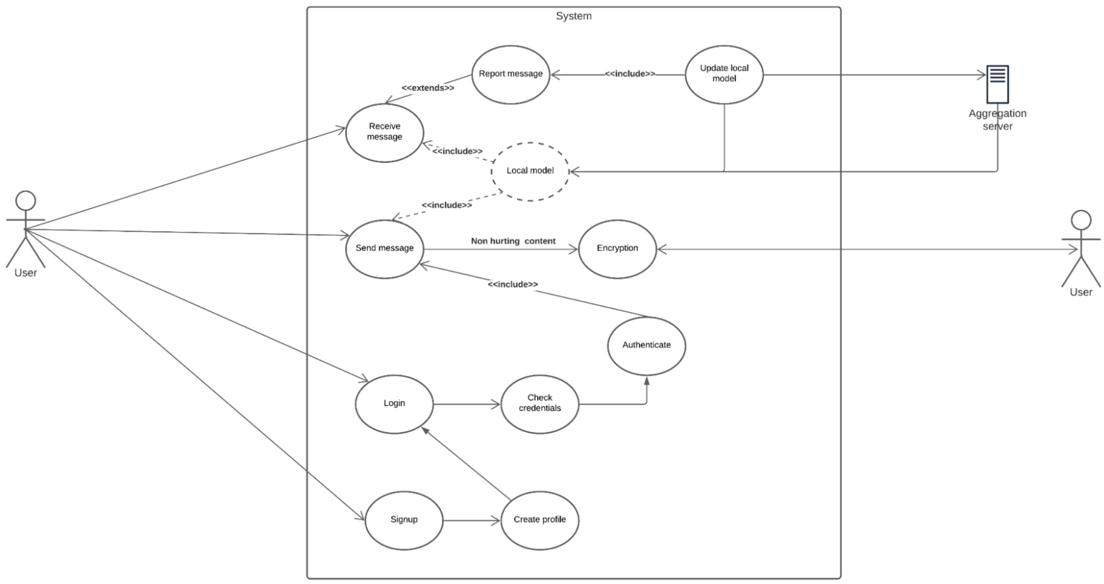

Figure 3.4: Use case diagram

Glossary
«include» : include tag is used when a process needs a helper function to carry out an operation.
«extend» : extend tag is used to represent a supplementary action.
i.e the functionality of the use case can be extended to a specific action if required. A simple analogy for both would be like, Consider sneezing. If you sneeze you are obviously going to close your eyes. Now it's good manners to say "excuse me" when sneezing. So in sneezing, «include» = close eyes «extend» = say "excuse me"

3.1.4 Data flow diagram

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. External entity, Process, Data store and Data flow are the major components of a data flow diagram.

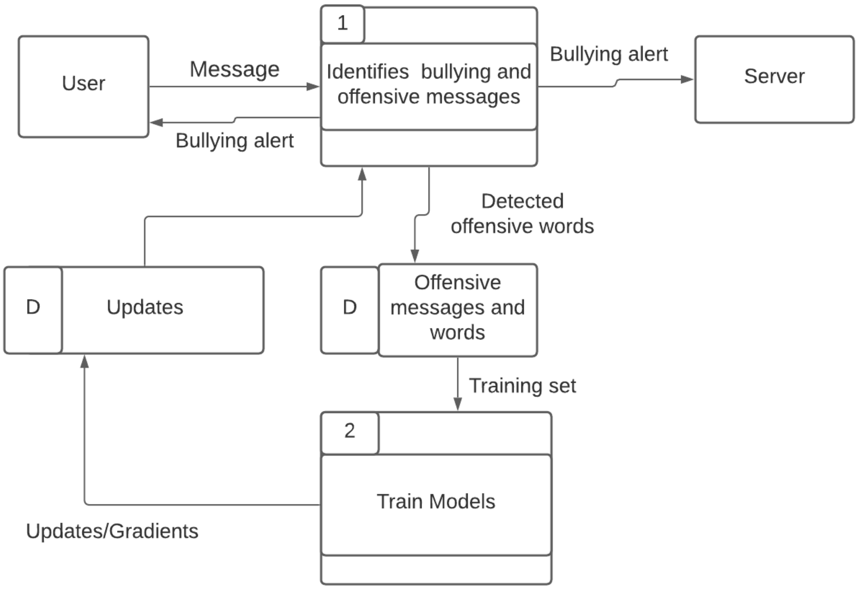

Figure 3.5: Data flow diagram

3.1.5 Entity Relationship diagram

ER diagram gives the structure of database used for the application. Figure 3.3 show ER diagram of our application. To make the application faster we store the hash of the message in a hash table, and its ER diagram is Figure 3.4

3.2 TEST CASE DESIGN

3.2.1 Requirement Based Testing

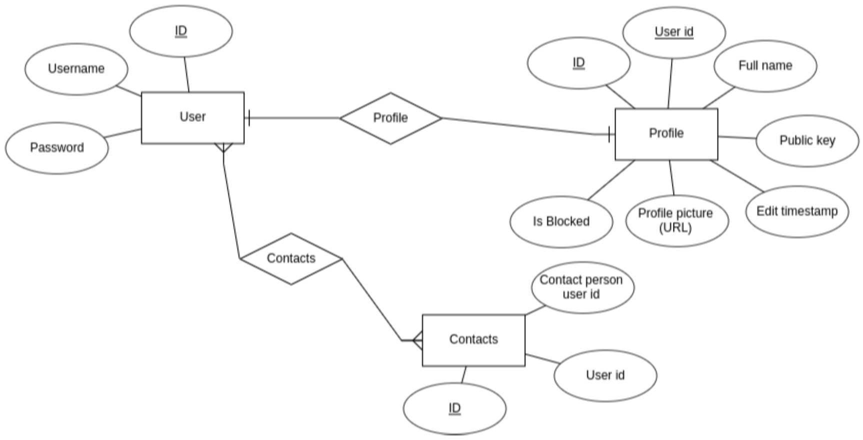

Figure 3.6: ER diagram of the application

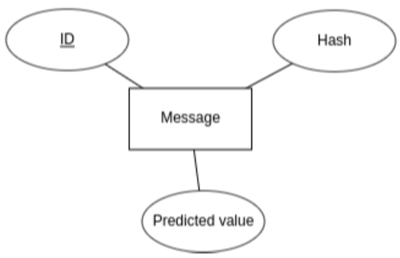

Figure 3.7: ER diagram of the text message

Table 3.1: Client React App

Test Priority	Module Name	Expected Result	Actual Result	Pass/Fail
High	Login	User must be able to login with username and password	Successfully logged in using username and password	Pass
Medium	Forgot password	Able to reset password with email id provided during registration	Password reset email sent to registered user	Pass
Low	Chat System	Display incoming and outgoing messages in respective columns	Working as per expectation	Pass
High	Send Message through Firebase API	Receiver must receive message	Working as per expectation	Pass
High	Encrypt Message using public key before sending	Encrypted text as output	Successfully encrypted the message	Pass
High	Decrypt Message using private key	Decrypted text as output	Successfully decrypted the text	Pass
High	Display Inference from model	Abusive or not	Working as per expectation	Pass
Medium	Registration API	Successfully register user	Working as per expectation	Pass
High	Public key private key generation	Generate keys in armoured format	Keys generated in required format	Pass
High	Public key private key storage	Store base64 keys	Working as per expectation	Pass
High	Public key private key fetch	Receive base 64 keys and decode	Working as per expectation	Pass

Table 3.2: Client Django App

Test Priority	Module Name	Expected Result	Actual Result	Pass/Fail
High	Report Message	Upon pressing the report option in the frontend,user should be able to take part in the federated learning process.	User successfully becomes a part of the federated learning process on clicking the report message.	Pass
High	Evaluate	When the messages are passed to this endpoint it should return the sentiment of the message (if its abusive or not)	On passing message the end point responds with the sentiment.	Pass
High	Private key storage	Receive POST request and store	Working as per expectection	Pass
High	Private key send	Send over GET request	Working as per expectation	Pass

Table 3.3: Authentication Server

Test Priority	Module Name	Expected Result	Actual Result	Pass/Fail
Low	Password Reset	User should be able to reset the password	User successfully able to reset the password	Pass
Low	Edit profile image	User should be able to edit the profile photo	User successfully able to edit the profile photo	Pass
Medium	Get public key using API	Receive public key using an identifier	Working as per expectation	Pass

Table 3.4: Federated Machine Learning

Test Priority	Module Name	Expected Result	Actual Result	Pass/Fail
Medium	Cross Validation of BERT Model	F1 score >= 0.91	Achived F1 score of 0.94 in federated model and 0.96 in initial model	Pass
High	Functioning of FL	Ensure the training accuracy and loss vary over rounds	Training accuracy and loss vary over rounds	Pass
High	Parameter passing	Pass initial parameters by converting them to compatible data type	Initial parameters convertered to suitable datatype before using them for federated learning	Pass

CHAPTER 4
CODING

4.1 FUNCTIONAL MODULES

4.1.1 Encrypting the message

Here we use public key encryption. So first we fetch the public key of the receiver from authentication server. Then the key is converted to armored format so that we can use it for encrypting the message. Next we encrypt the message using this key and output will be encrypted text in OpenPGP armored text format.

```
const encrypt_data = async (en_text) => {
    let response = await postData(`${config.SERVER_BASE_URL}/get_pub/`, {
        userId: Number(localStorage.getItem("receiverID")),
    });
    if (response.success === true) {
        let pub_key = Buffer.from(
            response.data.pub_key,
            "base64"
        ).toString();
        const publicKey = await openpgp.readKey({
            armoredKey: pub_key,
        });

        let encrypted = await openpgp.encrypt({
            message: await openpgp.createMessage({ text: en_text }),
            encryptionKeys: publicKey,
        });

        return {
            success: true,
            message: encrypted,
        };
    } else {
        return {
            success: false,
            message: "",
        };
    }
};
```

Figure 4.1: Encrypting the text message

4.1.2 Decrypting the message

Decryption follows the same format as that of encryption. First we fetch the private key from the local server and it is decrypted using a passphrase that we set up during registration. Using this key we decrypt the message and send it for inference.

```javascript
function decrypt_data(text, timestamp, receiverID) {

    postData(`${config.SERVER_BASE_URL}/get-private-key/`, {
        userId: receiverID,
    }).then((response) => {
        if (response.success === true) {
            const privateKey2 = Buffer.from(
                response.data.pri_key,
                "base64"
            ).toString();

            async function decrypt(enc_msg, privateKey) {
                const decrypted = await openpgp.decrypt({
                    message: await openpgp.readMessage({
                        armoredMessage: enc_msg, // parse armored message
                    }),
                    decryptionKeys: await openpgp.decryptKey({
                        privateKey: await openpgp.readPrivateKey({
                            armoredKey: privateKey2,
                        }),
                        passphrase: passphrase,
                    }),
                });
                return decrypted.data;
            }

            decrypt(text).then((value) => {
                //Inference and replace the value inside
                postData(`${config.CLIENT_APP_DJANGO}/evaluate/`, {
                    messages: [value],
                }).then((response) => {
                    if (response.success === true) {
                        document.getElementById(timestamp).innerText =
                            response.predictions[0] === 0
                                ? value
                                : "Abusive Content";
                    }
                });
            });
```

Figure 4.2: Decrypting the text message

4.1.3 Dataset and BERT Model

(i) Dataset

Twitter tweet dataset ENCASEH2020[1] [3]Founta AM et al, with 99996 data points. This particular dataset has three features: text, label and votes. There are 4 different types of labels: spam, abusive, normal and hateful.

[1] `https://github.com/ENCASEH2020/hatespeech-twitter`

(ii) Exploratory Data Analysis on dataset

Refer Figure 4.3 and Figure 4.4

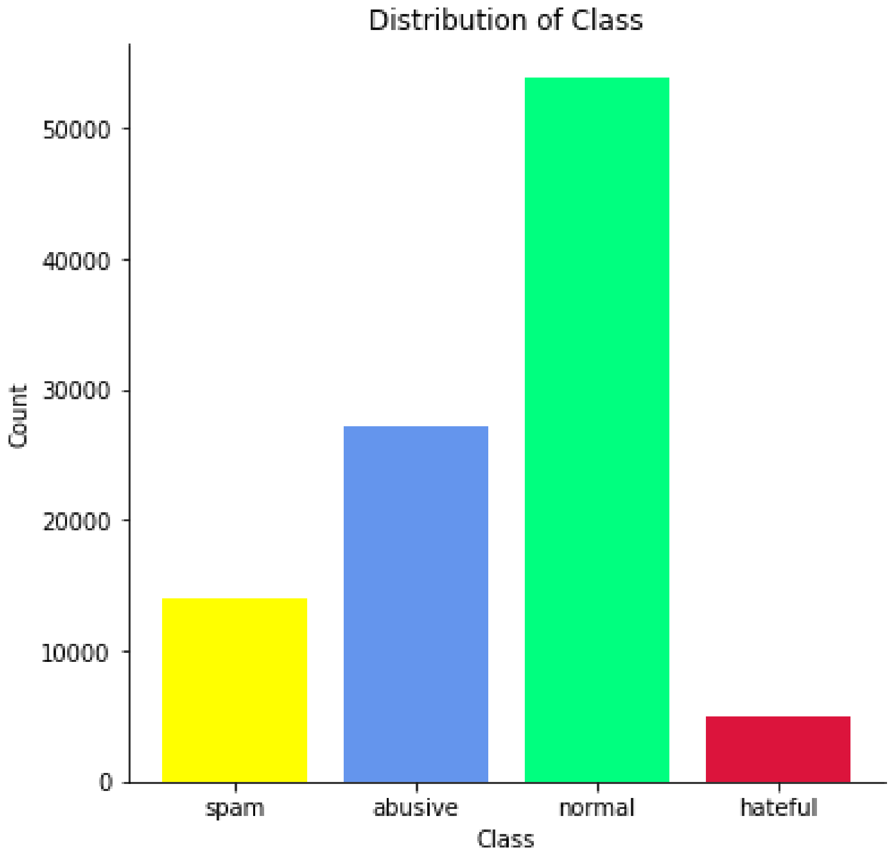

Figure 4.3: Distribution of data is as shown in the figure. We are interested only in abusive and normal classes. In order to make the dataset balanced. We will sample normal classes to the size of abusive classes.

(iii) Pre-processing Pipeline

Before passing value to the BERT model either for training or inference we have to pass the sentences through this pipeline

1. **URL Parsing:** First we have to replace URLs by ‘url’ keywords. This will ensure that model will see only the vectorizable terms.

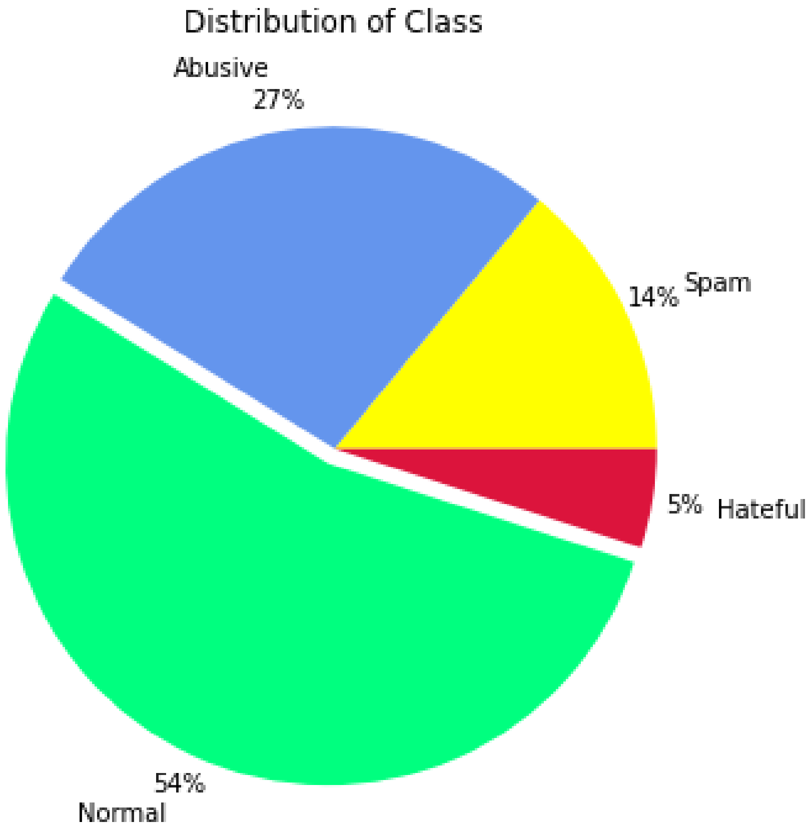

Figure 4.4: Percentage distribution of labels

2. **Removing Punctuations:** We don't want unwanted punctuations in the vectorized string.

3. **Replace Username:** Username does not add any meaning to the sentence, so we replace it with the 'userId' keyword.

4. **Remove RT Stamp:** Some of the tweets will have RT Stamp attached to it, since it is a reply text to some post.

5. **Hashtag to Text:** Hashtags can add additional meaning to the sentence, even though the sentence is not abusive. We convert hashtags to string using a python

library available as open source.

6. **Emoji to Text:** Meaning of the sentence can be hidden in emoji that makes the difference in meaning of the sentence. So we convert emoji to text using the python library.

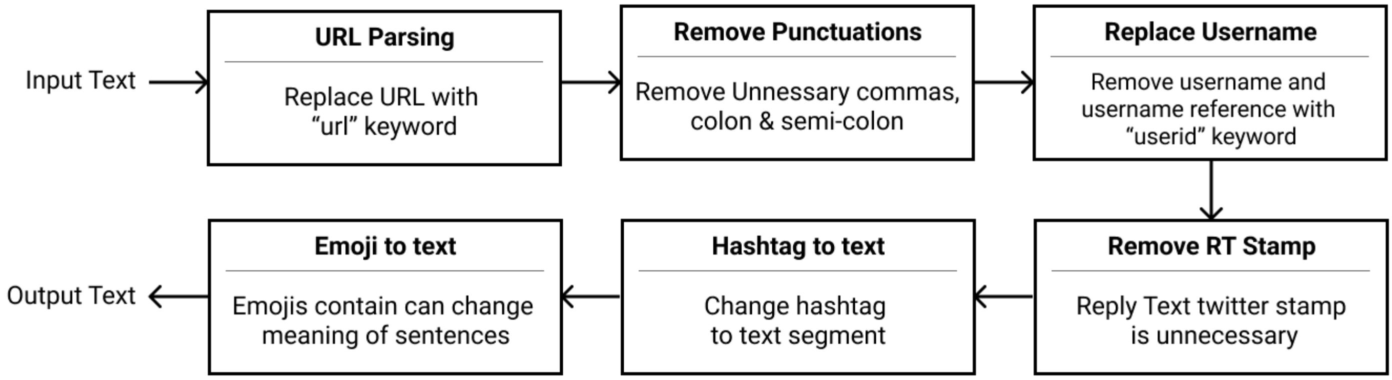

Figure 4.5: Pre-processing pipeline

(iv) Model Architecture

We are using the BERT model for the inference. Standard BERT has 12 transformer blocks, 768 hidden units and 12 self attention heads, making the model 389.71MB in size, heavy for edge device computing. But we can use a modified version of BERT ie. BERT Tiny has only 2 transformer blocks, 128 hidden units and 2 self attention heads that makes the model good for devices having less resources. Model size is 15.97MB.

(v) Training Setup

1. **Initial Model** Model is trained on 54300 data points. ie. 27150 of normal and abusive. Adam Weight Decay optimizer works well on transformer architecture. We set a learning rate of 0.001, and Cross Entropy as the loss function.

2. **Federated Model** The Federated Model is characterized mainly by the minimum number of participants and by the number of rounds for which the learning takes place. For testing our Federated Learning model implementing the same architecture as that of our initial model, we had initialized both the minimum number of clients and the number of rounds to a bare minimum value of 2. In addition the local model at the client side had accepted one text message at a time from the user for training. With this setup, we had witnessed the accuracy and loss changing across the rounds, though not significantly, confirming that the learning process is taking place. With sufficient set up and processing capabilities we expect to demonstrate the learning process and increase in accuracy on a large scale.

(vi) Evaluation of model

1. **Training vs Validation Accuracy** We have achieved an accuracy of 96% on average. Training and validation accuracy is shown below in Figure 4.7. Considering the scale of the graph it can be noted that the model is not overfitting to an extent.

2. **Confusion Metrics** The Confusion metric for the model is shown below. We can see that only 227 data points are misclassified compared to 5223 points.

3. **Classification Report** Model trained using the above setting has achieved an F1 Score of 0.96 against 0.91 in base paper [2]M. Behzadi et al. This is for the initial model. For the Federated model F1 score is 0.94.

4.1.4 Federated Learning

(i) Classifier Model

The Classifier Model is realized through transfer learning by concatenating a transformer with custom fully connected layers. The model takes the text as input and predicts the sentiment of the text whether it is abusive or not. Code is shown in Figure 4.10

(ii) Federated Client

The Federated Client module is responsible for performing client-level computation/training in a Federated Learning round. The code snippet illustrating Federated client is given in Figure 4.11. The start function creates an instance of the model and loads the data for training from the message parameter that it accepts. It has four methods namely

- `set_parameters()`
- `get_parameters()`
- `fit()`
- `evaluate()`

that interacts with the server module to pass on the parameter updates and get the global model updated. The client module is triggered when the user clicks the 'report-message' button in the chat application.

(iii) Federated Server

The federated server is responsible for accepting parameter updates from individual client models and aggregate the parameter changes. The configuration for the Federated server has been illustrated in Figure 4.12. Federated server uses FedAvg strategy with the number of rounds set to two. Additionally, a minimum of two clients are required for the Federated learning to start. The initial parameters are evaluated before the Federated Learning rounds as well as after every round.

(iv) Inference

The Evaluate method is responsible for returning the sentiment of the text in the chat section. The actual prediction of the sentiment is handled by the `predict_sentiment` function in Figure 4.14. Once the prediction is done it is the messages are hashed using MD5 and memoized. This ensures fast access to sentiment of the text on consecutive retrievals. Code is shown in Figure 4.13

4.1.5 Authentication Server

(i) Register User

Every user must be registered to the authentication server to enable chat service. During registration, username, password, full name and public key generated from the client side is stored in the database. Code section providing this API is documented below in Figure 4.15. API endpoint and usage is provided in the API documentation section. Data communication from and to the server is done using HTTP protocol and JSON formatted body section.

(ii) Search Users

For enabling search functionality in chat applications, we have to send user details to the client side. Not all the user details are sent, only those who are not blocked and people who are not in the user's contact list. Line 86 does the querying from the database. Code is shown in Figure 4.16

(iii) Public key distribution

Chat application is end-to-end encrypted, public key must be distributed to users who wish to communicate. Encryption from the sender side is done using the public key of the receiver. For this we have to store and provide public keys from the authentication server. Following code snippet serves user with the public key corresponding to the user id requested. Code is shown in Figure 4.17

(iv) User listing and user profile data

In the chat application, you can see a contact list navigation bar. To have that functionality, first we have to get the contact list corresponding to the user signed in and the user details corresponding to the contact user id. Code snippet is as follows in Figure 4.18. We have divided functions into two parts, this is to increase the modularity of the program. Line 125-143 will provide the list of users in the contact list. On receiving the list the client must iterate through each list and access user details through the API responsible for calling the second function i.e. line 145-162. The same function can be used for providing data for profile pages. This serves two purposes in one function.

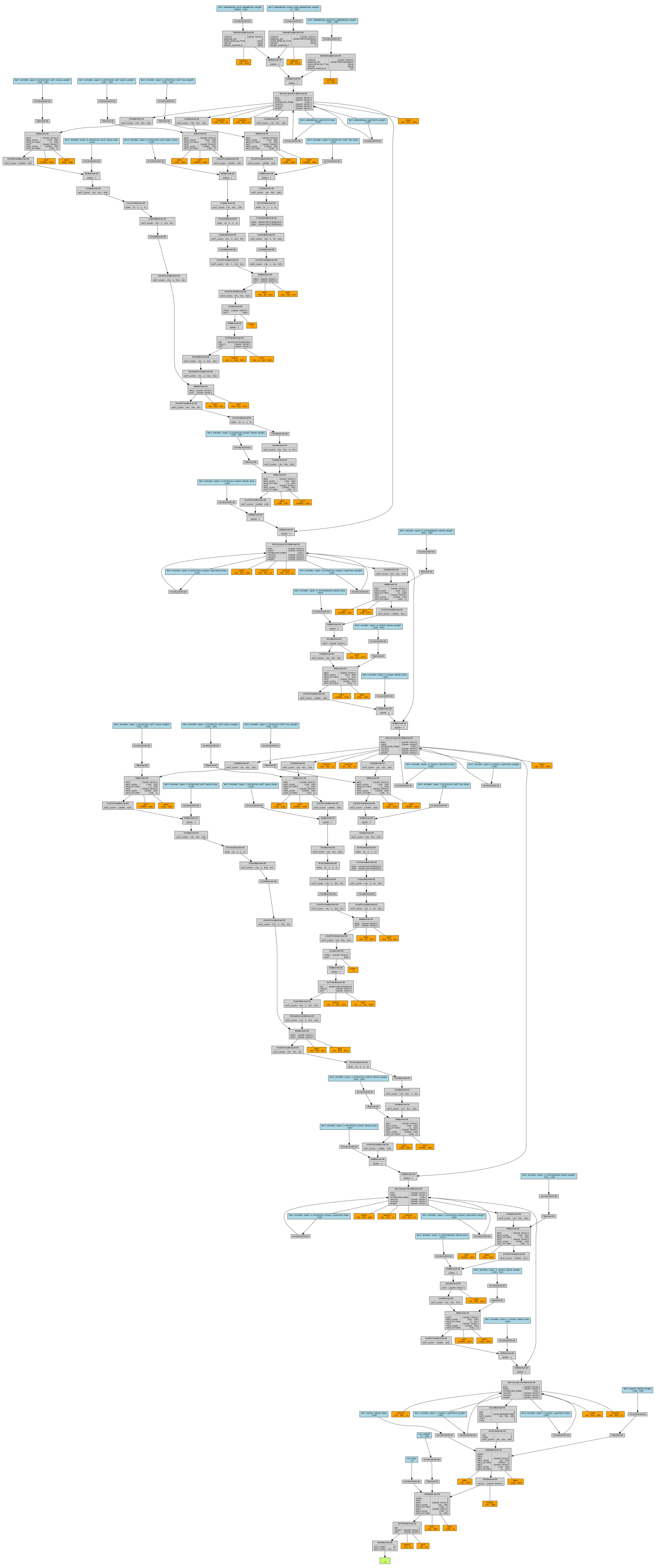

Figure 4.6: BERT Architecture

Figure 4.7: Training vs. Validation Accuracy

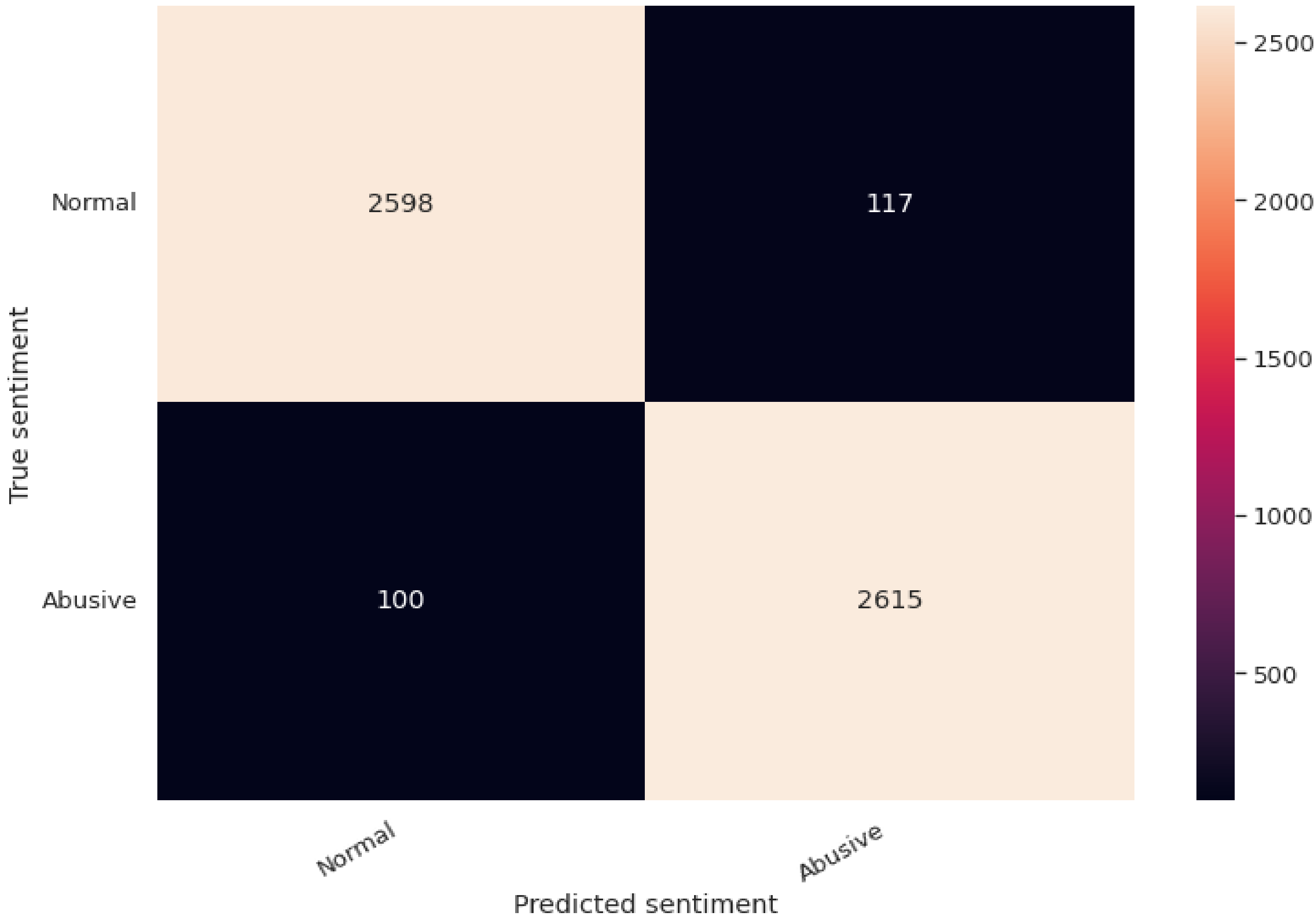

Figure 4.8: Confusion Matrix

	precision	recall	f1-score	support
Normal	0.96	0.96	0.96	2715
Abusive	0.96	0.96	0.96	2715
accuracy			0.96	5430
macro avg	0.96	0.96	0.96	5430
weighted avg	0.96	0.96	0.96	5430

Figure 4.9: Evaluation Report

```python
from transformers import BertModel
from torch import nn
CHECKPOINT = 'nreimers/BERT-Tiny_L-2_H-128_A-2'  # transformer model checkpoint


class CyberbullyingClassifier(nn.Module):

  def __init__(self, n_classes):
    super(CyberbullyingClassifier, self).__init__()
    self.bert = BertModel.from_pretrained(CHECKPOINT)
    # self.drop = nn.Dropout(p=0.3)
    self.out = nn.Linear(self.bert.config.hidden_size, n_classes)

  def forward(self, input_ids, attention_mask):
    bert_out = self.bert(
      input_ids=input_ids,
      attention_mask=attention_mask
    )
    pooled_output = bert_out[1]
    # output = self.drop(pooled_output)
    return self.out(pooled_output)
```

Figure 4.10: Classifier Model Code

```python
def start(message):
    net = CyberbullyingClassifier(n_classes = 2).to(DEVICE)
    # trainloader, testloader, train_count, test_count = load_data(message)
    trainloader, train_count = load_data(message)

    # Flower client
    class FedClient(fl.client.NumPyClient):
        def get_parameters(self):
            return [val.cpu().numpy() for _, val in net.state_dict().items()]

        def set_parameters(self, parameters):
            params_dict = zip(net.state_dict().keys(), parameters)
            state_dict = OrderedDict({k: torch.Tensor(v)
                                     for k, v in params_dict})
            print("Setting parameters...")
            net.load_state_dict(state_dict, strict=True)
            print("Done setting parameters.")

        def fit(self, parameters, config):
            self.set_parameters(parameters)
            print("Training Started...")
            accuracy, loss = train(net, trainloader, epochs=1, n_examples=train_count)
            print("Training Finished.")
            print("Saving Weights...")
            torch.save(net.state_dict(), PATH)
            return self.get_parameters(), len(trainloader), {"accuracy": float(accuracy)}

        def evaluate(self, parameters, config):
            print("[info] Client side evaluation not implemented.")
            # self.set_parameters(parameters)
            # accuracy, loss = test(net, testloader, n_examples=test_count)
            # return float(loss), len(testloader), {"accuracy": float(accuracy)}

    # Start client
    fl.client.start_numpy_client("[::]:9999", client=FedClient())
```

Figure 4.11: Federated Client Code

```python
if __name__ == "__main__":
    def get_eval_fn(model):
        """Return an evaluation function for server-side evaluation."""

        # Load data and model here to avoid the overhead of doing it in `evaluate` itself
        processed_df = pd.read_csv('processed_hatespeech_text_label.csv')
        processed_df.text = processed_df.text.apply(clean_text)
        test_df = processed_df.iloc[int(len(processed_df) * 0.995):]
        tokenizer = BertTokenizer.from_pretrained(CHECKPOINT)
        testloader = create_data_loader(test_df, tokenizer, MAX_LEN, BATCH_SIZE)


        # The `evaluate` function will be called after every round
        def evaluate(weights: fl.common.Weights) -> Optional[Tuple[float, float]]:
            params_dict = zip(model.state_dict().keys(), weights)
            state_dict = OrderedDict({k: torch.tensor(v) for k, v in params_dict})
            model.load_state_dict(state_dict)  # Update model with the latest parameters
            accuracy, loss = test(model, testloader, len(test_df))
            return loss, {"accuracy":accuracy}

        return evaluate

    # Load the model
    model = CyberbullyingClassifier(n_classes = 2)

    weights = torch.load("best_model_state.bin", map_location=torch.device('cpu'))
    np_weights = [v.numpy() for _, v in weights.items()]
    parameters = fl.common.weights_to_parameters(np_weights)

    # Define strategy
    strategy = fl.server.strategy.FedAvg(
        fraction_fit=1.0,
        fraction_eval=1.0,
        min_available_clients=2,
        initial_parameters=parameters,
        eval_fn=get_eval_fn(model),
    )

    # Start server
    fl.server.start_server(
        server_address="[::]:9999",
        config={"num_rounds": 2},
        strategy=strategy,
    )
```

Figure 4.12: Federated Server Code

```python
@csrf_exempt
def evaluate(request):
    if request.method == 'POST':
        body = request.body.decode('utf-8')
        content = json.loads(body)
        messages = content["messages"]
        if type(messages)!=list:
            print("Verified")
            return JsonResponse({'success':False, 'message':'Expected list of messages. Recieved {}'.format(type(message)) })
        predictions = []
        for message in messages:
            hashmessage = hashlib.md5(message.encode())
            if Message.objects.filter(hash=hashmessage.hexdigest()).exists():
                predictions.append(1 if Message.objects.filter(hash=hashmessage.hexdigest())[0].sentiment else 0)
            else:
                try:
                    pred = predict_sentiment(message)
                    predictions.append(pred)
                except:
                    return JsonResponse({'success':False,
                     'message':'Error in prediction. Please try again or make sure your Internet connection is on. ' })
                try:
                    Message.objects.create(hash=hashmessage.hexdigest(), sentiment=pred)
                except:
                    return JsonResponse({'success':False, 'message':'Error in saving message.' })

        return JsonResponse({'success':True, 'predictions':predictions })
    else:
        return JsonResponse({'success':False, 'predictions':"Invalid Request method" })

```

Figure 4.13: Message Evaluation Code

```python
def predict_sentiment(text):
    CHECKPOINT = 'nreimers/BERT-Tiny_L-2_H-128_A-2'
    DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    BATCH_SIZE = 32
    MAX_LEN = "max_length"
    CLASS = ['normal', 'abusive']

    tokenizer = BertTokenizer.from_pretrained(CHECKPOINT)
    encoding = tokenizer(
                text, padding=MAX_LEN, truncation=True, return_tensors="pt")
    net = CyberbullyingClassifier(2)
    net.load_state_dict(torch.load("fed_client/weights/final_weights.pt"))
    net.eval()
    input_ids = encoding["input_ids"].to(DEVICE)
    attention_mask = encoding["attention_mask"].to(DEVICE)
    with torch.no_grad():
        output = net(
                    input_ids=input_ids,
                    attention_mask=attention_mask
                )
    _, preds = torch.max(output, dim=1)
    return int(preds[0])
```

Figure 4.14: Predict Sentiment Code

```python
@csrf_exempt
def register(request):
    if request.method == "POST":
        body = request.body.decode('utf-8')
        content = json.loads(body)
        username = content["username"]
        password = content["password"]
        fullname = content["fullname"]
        public_key = content["publickey"]
        if username == "" or password =="":
            response = {"success":False,"message":"Please verify the inputs"}
        else:
            user_data_exist = User.objects.filter(username=username).count()
            if user_data_exist == 0:
                user_data = User.objects.create_user(
                    username=username,
                    password=password
                )
                user_profile = Profile.objects.get(uid=user_data.id)
                user_profile.fullname = fullname
                user_profile.public_key = public_key
                user_profile.save()
                response = {"success":True,"data":{"userId": user_data.id}}
            else:
                response = {"success":False,"message":"Please choose another username"}
    else:
        response = {"success":False,"message":"Invalid request method. Expected a POST request"}

    return JsonResponse(response)
```

Figure 4.15: Register User Code

```python
@csrf_exempt
def get_user_list(request):
    if request.method == "POST":
        body = request.body.decode('utf-8')
        content = json.loads(body)
        user = content['user']
        ids = []
        contacts = Contacts.objects.filter(user__id=user)
        for i in contacts:
            ids.append(i.contact.id)

        # Avoid showing users in his contact and himself
        avoid_ids = ids + [user]
        profiles = Profile.objects.filter(isBlocked=False).filter(~Q(uid__id__in=avoid_ids))
        users = []
        for i in profiles:
            users.append({
                "name": i.fullname,
                "profile_picture": i.profile_picture.url,
                "uid": i.uid.id
            })
        response = {"success":True, "users": users}

    else:
        response = {"success":False, "message": "Cannot fetch data"}

    return JsonResponse(response)
```

Figure 4.16: Search User Code

```python

@csrf_exempt
def get_pub(request):
    if request.method == "POST":
        body = request.body.decode('utf-8')
        content = json.loads(body)
        id = int(content["userId"])

        try:
            isBlocked = Profile.objects.get(uid=id).isBlocked
            if isBlocked:
                response = {"success":False,"message":"Invalid userId"}
            else:
                pub_key = Profile.objects.get(uid=id).public_key
                response = {"success":True,"data":{"pub_key":pub_key}}
        except Profile.DoesNotExist:
            response = {"success":False,"message":"Invalid userId"}
    else:
        response = {"success":False,"message":"Invalid request method. Expected a POST request"}

    return JsonResponse(response)
```

Figure 4.17: Public Key Distribution Code

```python
@csrf_exempt
def get_contact_list(request):
    if request.method == "POST":
        body = request.body.decode('utf-8')
        content = json.loads(body)
        id = content['id']

        data = Contacts.objects.filter(user=id)
        contact_ids = []
        for i in data:
            contact_ids.append(i.contact.id)


        response = {"success": True, "contacts": contact_ids}

    else:
        response = {"success": False, "message": "False HTTP Method"}

    return JsonResponse(response)

@csrf_exempt
def get_user_data(request):
    if request.method == "POST":
        body = request.body.decode('utf-8')
        content = json.loads(body)
        id = content['id']
        try:
            user_profile = Profile.objects.get(uid=id)
            response = {"success": True,"data":{
                "fullname": user_profile.fullname,
                "picture": user_profile.profile_picture.url
            }}
        except Profile.DoesNotExist:
            response = {"success":False,"message":"No user"}
    else:
        response = {"success":False,"message":"Invalid request method. Expected a POST request"}

    return JsonResponse(response)
```

Figure 4.18: Contact List Code

CHAPTER 5
PERFORMANCE EVALUATION

Section 5.1 gives an overview of the testing environment through different perspectives. Section 5.2 gives the various results of the testing process.

5.1 EVALUATION SCENARIO

5.1.1 Scenario based testing

For obtaining a real-world scenario while considering the technical limitation, the application was given to two users in our college. The users typed in both regular and offensive texts. Now the same two users were made to participate in the Federated Learning process and the tests were conducted again. The backend was constantly monitored to check if any crash occurred.

5.1.2 Load Testing

The load test was conducted by sharing the application with 4 users and according to [8]Beutel et al, Flower framework was tested with 15M users. The application performed smoothly, without any effect on the latency of APIs. Except for the initial time of inference, which we managed to bring down using memoization, the application rendered a user-friendly experience.

5.1.3 Install/Uninstall testing

Installing the dependencies to run the requires a single command. On average, it takes 81.08 seconds for all the dependencies to get installed and the application to be fully-functional.

5.2 RESULTS

5.2.1 Initial Model

Figure 5.1 and Figure 5.2 illustrates the performance of the initial model on CPU and GPU. More details about the initial model can be found in Chapter 4.

Figure 5.1: Performance (CPU)

Figure 5.2: Performance (GPU)

5.2.2 Federated Learning

Table 5.1 summarises the performance of the Federated Learning across two rounds. During each round the model was trained on an abusive text reported by the user. A behaviour to notice is the how the accuracy decreases and loss increases across 2 rounds. This, although not desired, is an affirmative indication that the learning happens seamlessly. The undesired results can be attributed to the very small data samples that the model sees at

	Accuracy	**Loss**	**Time (s)**
Initialisation	0.94485	0.16244	-
Round 1	0.94117	0.17697	527.358
Round 2	0.94117	0.19437	567.002

Table 5.1: Federated learning performance

a time while training. This is similar to the case of Stochastic Gradient Descent (SGD) where trend may be desired although the acute variation may be undesired as in Figure 5.3.

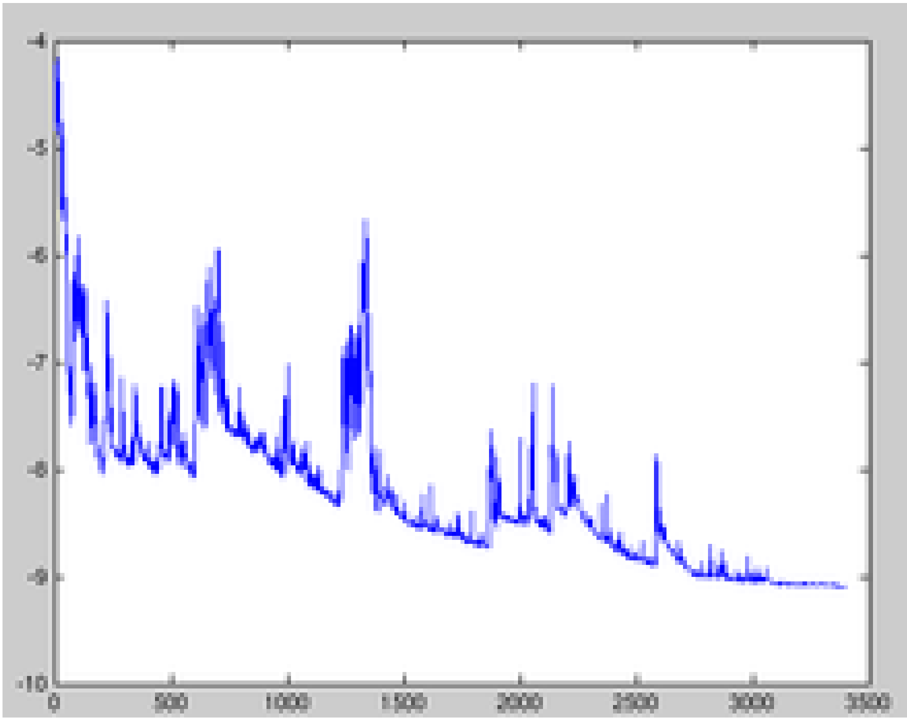

Figure 5.3: Fluctuating loss of a random learning using SGD

CHAPTER 6
DOCUMENTATION

6.1 INTRODUCTION

E2EE Messaging System enhanced with federated machine learning for cyberbullying is called in short as *Schat*. Proposed system is developed using two coding languages: Python, and JavaScript. Detailed system architecture and working is explained in the above section. This system was designed for cross platform usage and tested on the Linux operating system , specifically Ubuntu 20.04 and Ubuntu 18.04; though it must work on Windows and Mac OS. We need not bother about the authentication server and model server which is running on Oracle cloud in a Linux virtual machine(Ubuntu 20.04). Installation and user manual of software is given below:

6.2 INSTALLATION

Prerequisite applications/programs:

1. Python 3.9
2. NodeJS v17.3.0 and npm package manager v8.3.0

User have to create a directory(folder) in desired location and clone the following repository:

1. `https://github.com/Cubemet/script.git`
2. `https://github.com/Cubemet/client-app.git`

3. `https://github.com/Cubemet/client-react-app.git`

Open the script repository in the terminal. Execute run.sh in bash. Initially it will install all the dependencies and initialize local databases in respective locations. After completion, rerun the same script which will start local processes and open the user interface in default browser automatically. Note that this script is only designed for the Linux operating system. Details are provided below:

6.2.1 Cloning Repository

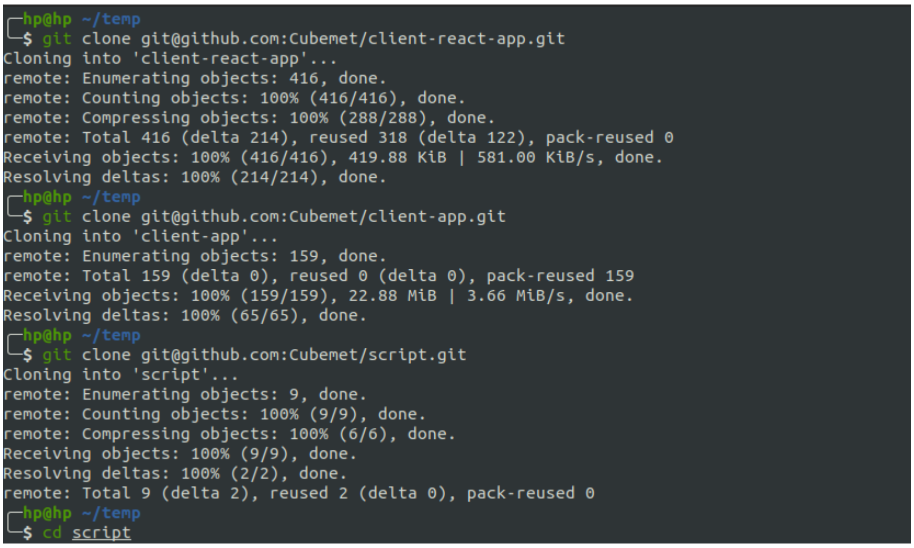

Figure 6.1: Clone source code from GitHub

6.2.2 Installing Dependencies and Database

Please refer Figure 6.2 to Figure 6.4.

6.2.3 Building React App

Please refer Figure 6.5 and Figure 6.6.

Figure 6.2: Running Script and installing python dependencies inside virtual environment

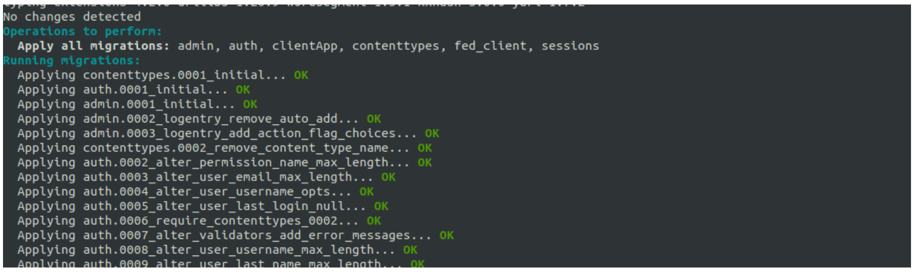

Figure 6.3: Creating local database

6.2.4 Launch Application

Re-run the same script to launch application in default browser. Please refer Figure 6.7.

6.3 WORKING WITH THE PRODUCT

6.3.1 Register user

New users can be registered by clicking register user in the login page. Just provide a unique username, full name and password. User details will be sent to the authentication server with a public key generated by your computer. Refer Figure 6.8.

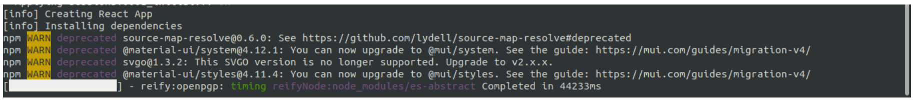

Figure 6.4: Installing JavaScript npm dependencies

Figure 6.5: Building React App

6.3.2 Login user

Users can login to their account using the credentials provided during the registration process. Refer Figure 6.9.

6.3.3 Home Screen

In the home screen you can see the navigation bar(Figure 6.10) in the top section. Search, user profile and logout functionalities are provided in the top left corner of the screen. Also users can opt to see abusive content by clicking the eye icon, and hide if they want using the same icon. Below that section, we can select the user to whom you have to chat. Active chat is shown in a different color than others. . Below the navigation bar, we can see the chat area (Figure 6.11). History of chat is displayed as a list. Most recent ones are the bottom most. Auto scroll is enabled to help users scroll to the end. Users can send messages using the text box and send button. Note that, abusive content messages are hidden from users by Abusive content messages.

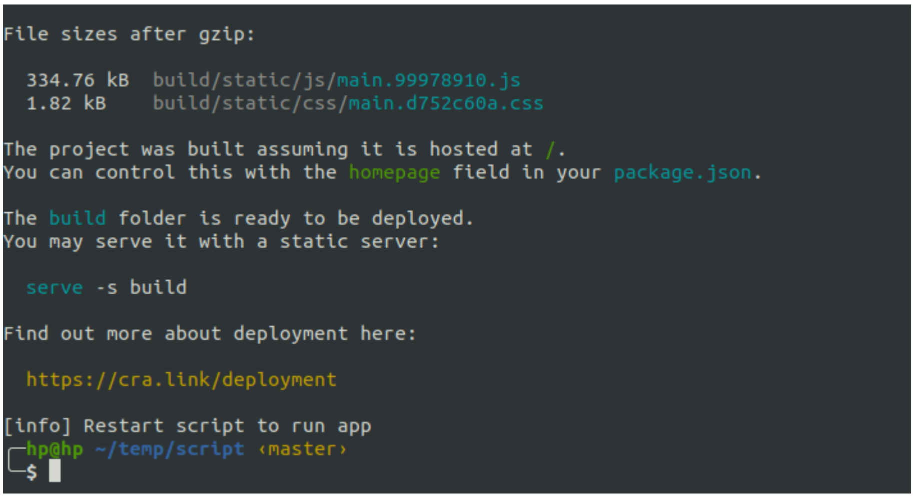

Figure 6.6: After building React App, restart script

Figure 6.7: Launch Application

6.3.4 Search user

People can search users using the search feature. Just click on the search button from the home page then type in the name of the person in the text box (Figure 6.12), as you type the names will be displayed below. If you click on the contact, that user will be added to your contact list. You cannot see users who are currently in your contact list.

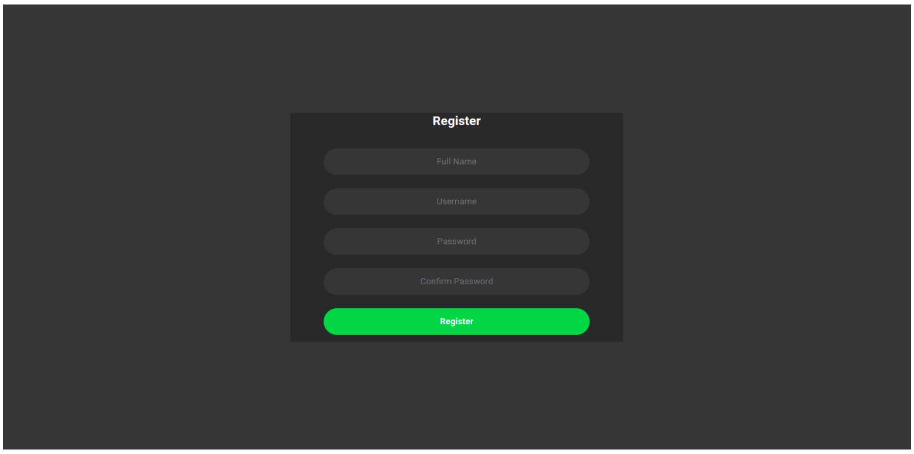

Figure 6.8: Register user

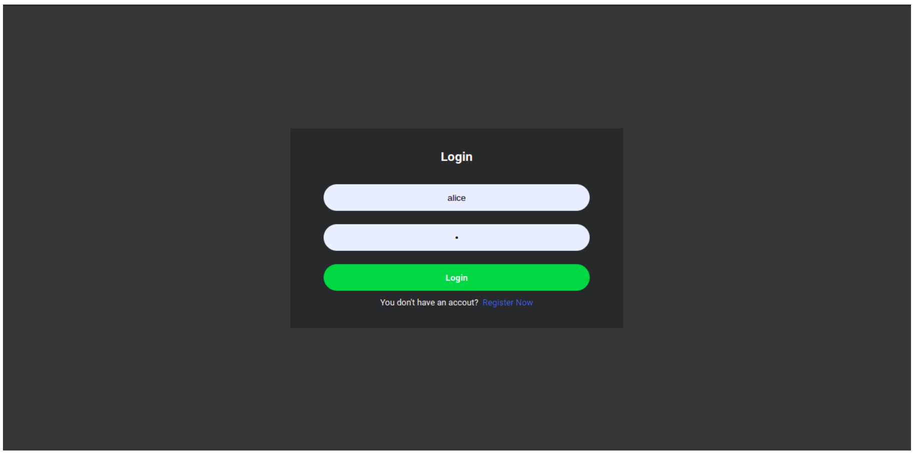

Figure 6.9: Login user

6.4 CONTACT

For queries and complaints, users can contact us through the team email id provided.

`cubemet.gec@gmail.com`

Figure 6.10: Navigation bar

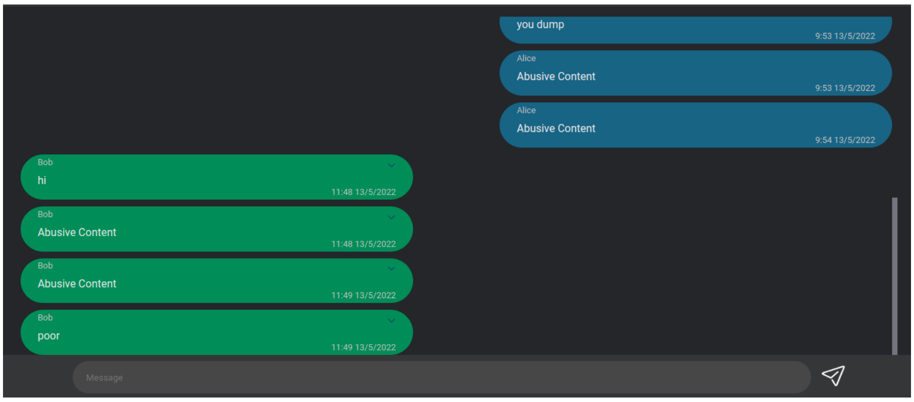

Figure 6.11: Chat Area

Figure 6.12: Search user

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

This project aims to create a user-friendly chat application that obscures unwanted and offensive text messages using a sentiment analysis model while conserving the privacy of the user using Federated Learning. Although some of the existing chat applications provide the user to report chats or messages, they train their model by agglomerating data on a single server. This poses the threat of privacy infiltration as user data is taken from their device for training the model. Federated Learning leverages the power of edge computing to train the model on users' data on their own device and aggregate the updates coming from individual devices on a central server. The system was implemented using the Flower framework in PyTorch and was tested and evaluated. The system can be a stepping stone to all privacy-preserved chat applications in the coming future.

7.2 ADVANTAGES

Following are the advantages of our application.

- Chats are end-to-end encrypted using Open PGP enabling secure communication among the users.
- User privacy is conserved since user data is not outsourced for training our model.
- It offers a user-friendly desktop application that conceals offensive messages making it suitable for use even among students.

7.3 LIMITATIONS & FUTURE EXPANSIONS

Following limitations were found in the current implementation of the system.

- The application only supports one-to-one chats presently. We plan to incorporate group communication and broadcast features in the future release of our application.
- The chat application is currently implemented as a web application. A major reason for such an initiative is the lack of production level SDKs. The support can be extended for Android devices as well after switching our existing Deep Learning framework from PyTorch to Tensorflow which provides Federated Learning support for Android devices as well.
- The system is currently characterized by large setup time and slow performance. We plan to bring down this time by merging the functionalities and reducing the use of multifaceted frameworks.

REFERENCES

[1]Li, T., Sahu, A. K., Talwalkar, A., and Smith, V., "Federated Learning: Challenges, Methods, and Future Directions", IEEE Signal Processing Magazine, vol. 37, no. 3, pp. 50–60, 2020. doi:10.1109/MSP.2020.2975749.

[2] M. Behzadi, I. G. Harris and A. Derakhshan, *"Rapid Cyber-bullying detection method using Compact BERT Models," 2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, 2021, pp. 199-202, doi: 10.1109/ICSC50631.2021.00042.

[3] Founta, A.M., Djouvas, C., Chatzakou, D., Leontiadis, I., Blackburn, J., Stringhini, G., Vakali, A., Sirivianos, M., & Kourtellis, N. (2018). Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior. In 11th International Conference on Web and Social Media, ICWSM 2018.

[4] Nilsson, Adrian et al. "A Performance Evaluation of Federated Learning Algorithms." Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning (2018): n. pag

[5] Devlin, J., Chang, M.-W., Lee, K.,an Toutanova, K. (2019). *BERT:Pre-training of Deep Bidirectional Transformers for Language Understanding*.

[6] Swanand Kadhe, Nived Rajaraman, O. Ozan Koyluoglu, & Kannan Ramchandran. (2020). FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning.

[7] M. S. Jere, T. Farnan and F. Koushanfar, "A Taxonomy of Attacks on Federated Learning," in IEEE Security & Privacy, vol. 19, no. 2, pp. 20-28, March-April

2021, doi: 10.1109/MSEC.2020.3039941.

[8] Beutel, Daniel J., Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, et al. 2020. "Flower: A Friendly Federated Learning Research Framework." arXiv [cs.LG]. arXiv. http://arxiv.org/abs/2007.14390.

APPENDIX – A
APPENDIX

A.1 API DOCUMENTATION

(i) Authentication Server

Refer table from A.1 to A.7

(ii) Client Application server

Refer table from A.8 to A.11

Table A.1: Auth-server : Register user

Name	Register User
Method	POST
Endpoint	/register/

Request body Usage (JSON)	{
	"username": <string>,
	"password": <string>,
	"fullname": <string>,
	"publickey": <string>
	}

Response body structure (JSON)	{
	"success": <bool>,
	"data": {
	"userId": <int>
	}
	}

Example Request	{
	"username":"kkowsikpai",
	"password":"a",
	"fullname": "Kowsik Nandagopan D",
	"publickey": "LS0tLS1CR. . . "
	}

Example Response	{
	"success": true,
	"data": {
	"userId": 25
	}
	}

Table A.2: Auth-server: Login

Name	Login
Method	POST
Endpoint	/login/

Request body Usage (JSON)	{
	"username": <string>,
	"password": <string>
	}

Response body structure (JSON)	{
	"success": <bool>,
	"data": {
	"userId": <int>
	}
	}

Example Request	{
	"username":"kowsikpai",
	"password":"a"
	}

Example Response	{
	"success": true,
	"data": {
	"userId": 25
	}
	}

Table A.3: Auth-server: Get public key

Name	Get Public key
Method	POST
Endpoint	/get_pub/

Request body Usage (JSON)	{
	"userId": <int>
	}

Response body structure (JSON)	{
	"success":<bool>,
	"data": {
	"pub_key": <string>
	}
	}

Example Request	{
	"userId": 20
	}

Example Response	{
	"success": true,
	"data": {
	"pub_key": "LS0tLS1CR..."
	}
	}

Table A.4: Auth-server: Get profile data

Name	Get Profile Data
Method	POST
Endpoint	/profile/

Request body Usage (JSON)	{
	"id": <int>
	}

Response body structure (JSON)	{
	"success":<bool>,
	"data": {
	"fullname": <string>,
	"picture": <string>
	}
	}

Example Request	{
	"id": 2
	}

Example Response	{
	"success": true,
	"data": {
	"fullname": "Kowsik Nandagopan D",
	"picture": "/media/default.png"
	}
	}

Table A.5: Auth-server: Get contact list

Name	Get Contact List
Method	POST
Endpoint	/get_contact_list/

Request body Usage (JSON)	{
	"id": <int>
	}

Response body structure (JSON)	{
	"success": <bool>,
	"contacts": <list>[<int>]
	}

Example Request	{
	"id": 2
	}

Example Response	{
	"success": true,
	"contacts": [20, 1, 22]
	}

Table A.6: Auth-server: Search suggestions

Name	Search Suggestion
Method	POST
Endpoint	/get_user_list/

Request body Usage (JSON)	{
	"user": <int>
	}

Response body structure (JSON)	{
	"success":<bool>,
	"users": [
	{
	"name": <string>,
	"profile_picture": <string>,
	"uid": <int>
	},
	]
	}

Example Request	{
	"user": 2
	}

Example Response	{
	"success": true,
	"users": [
	{
	"name": "Root",
	"profile_picture": "/media/default.png",
	"uid": 6
	},
	]
	}

Table A.7: Auth-server: Add contacts

Name	Add Contact
Method	POST
Endpoint	/add_contact_item/

Request body Usage (JSON)	{
	"user":<int>,
	"id": <int>// Contact Person user id
	}

Response body structure (JSON)	{
	"success":<bool>,
	"message": <string>
	}

Example Request	{
	"user": 21,
	"id": 6
	}

Example Response	{
	"success": true,
	"message": "Contact added"
	}

Table A.8: Client-App: Model Evaluation

Name	Model Evaluation
Method	POST
Endpoint	/evaluate/

Request body Usage (JSON)	{
	"messages": <list>[<string>]
	}

Response body structure (JSON)	{
	"success":<bool>,
	"predictions": <list>[<int>]
	}

Example Request	{
	"messages": [“hi"]
	}

Example Response	{
	"success": true,
	"predictions": [0]
	}

Table A.9: Client-App: Report Message (Trigger Training)

Name	Report Message
Method	POST
Endpoint	/report-message/

Request body Usage (JSON)	{
	"message": <string>
	}

Response body structure (JSON)	{
	"status”: <string>
	}

Example Request	{
	"messages": “*****”
	}

Example Response	{
	"status”: “Message Reported”
	}

Table A.10: Client-App: Save Private Key

Name	Save Private Key
Method	POST
Endpoint	/private-key/

Request body Usage (JSON)	{
	"pkey": <string>
	}

Response body structure (JSON)	{
	"success": <bool>,
	"message": <string>
	}

Example Request	{
	"pkey": “qwerty”
	}

Example Response	{
	"success": true,
	"message": "Private key saved successfully."
	}

Table A.11: Client-App: Get private key

Name	Get Private Key
Method	GET
Endpoint	/private-key/

Response body structure (JSON)	{
	"success": <bool>,
	"pkey": <string>
	}

Example Response	{
	"success": true,
	"pkey": "qwerty"
	}