Flagging Hate Speech and Cyber-Bullying in Private Chats

Using distributed machine learning and federated learning techniques to preserve privacy when learning from personal data

Aditya Srivastava
CSCI 6502-001
University of Colorado, Boulder
aditya.srivastava@coloradu.edu

Harsh Gupta
CSCI 6502-001
University of Colorado, Boulder
harsh.gupta@coloradu.edu

ABSTRACT

Online social networks and chat services have rampant issues of hate speech and cyber-bullying, which cause emotional harm, incite violence, and contribute to a toxic online environment. Service providers such as Twitter, Facebook, Reddit and others try to curb such behaviors by flagging offensive content (often using real people in the loop acting as content moderators) and warning users about it. While content on social networks is often open and public, such manual moderation cannot be performed on personal chats (such as on WhatsApp, Messenger, and texts) as people value their privacy when utilizing these channels, and would not want a third party to be privy to their conversations.

In this report we describe a system that utilizes distributed machine learning and federated learning techniques in the cloud to tag offensive content in personal chats, with none of the private data leaving the edge devices belonging to the individual users. This allows users to preserve the privacy of their data while taking a positive step in the direction of protecting them from hate speech and cyber-bullying.

CCS CONCEPTS

• Federated Learning • Distributed Computing • Privacy • Cloud Computing

KEYWORDS

Federated Learning, Machine Learning, Distributed Computing, Privacy, Cloud Computing

1 Introduction

The UN Strategy and Plan of Action on Hate Speech defines hate speech as…“any kind of communication in speech, writing or behavior, that attacks or uses pejorative or discriminatory language with reference to a person or a group on the basis of who they are, in other words, based on their religion, ethnicity, nationality, race, color, descent, gender or other identity factor.” [1]

Online cyber-bullying is described as “bullying with the use of digital technologies,” [7] and is prevalent on social media and instant messaging services. Unfortunately, unlike face-to-face bullying, cyberbullying leaves a digital footprint, lending it permanence.

Most social media companies take a strong stance against hateful content, and try to police their platforms to prevent users from engaging in such acts. [8,9] Many services employ real people to act as content moderators [10], and many use automated tools to do so. [11] Unfortunately running such moderation mechanisms is not possible on data that users want to keep private, since such data cannot be exposed to third party human moderators, and cannot be used as training data for machine learning models without violating the privacy of said users.

Federated learning is a machine learning paradigm that allows training data to remain decentralized and private, and learning is performed via locally-computed updates. [12] Since the only elements transmitted between the client and the server are the finetuned weights (or gradients, depending on the approach employed), and since the original data cannot be reconstructed from the weights (or gradients), the data stays private while allowing machine learning algorithms to be trained on it.

Thus, federated learning is a good solution for training automated content moderation tools without exposing users’ private data. The technology also fits the context of this course due to its emphasis on distributed computing on edge devices, and its ability to harness massive amounts of data.

2 Related Work

The concept of federated learning was first introduced by McMahan et al. (2017) [12] and described a method for distributed training and then unified aggregation of ML models. The accumulation algorithm in the paper was simple model averaging, i.e. the weights of the global model after each cycle were updated by averaging the weights of each of the finetuned client models.

The concept has since been applied to various domains [15], such as in mobile devices for next word prediction [13], in healthcare for personal monitoring IoT devices [14], and in finance for automated credit scoring. [16]

There has also been work on productionizing federated learning systems at scale. Google published its work detailing the high level design of their federated learning architecture geared towards Android devices. [17] Their architecture employs coordinator nodes (in addition to master and slave nodes) which coordinate learning and aggregation cycles based on availability. Microsoft Research also released a framework geared towards the research and development of high performance federated learning systems. [18]

3 System Description

3.1 Dataset

In keeping with the spirit of the course, we wanted to test our model on the largest possible amount of data. Thus, we combined multiple hate speech and cyber-bullying datasets to evaluate learning capability. On combining and preprocessing the datasets we found many duplicates between them, which were removed. The datasets chosen are listed in the table below, along with their respective sample sizes.

Dataset	Offensive	Inoffensive	Sample Size
Hate Speech Dataset [2]	599	2399	2998
Cyberbullying dataset [3]	57581	390552	448133
ConvAbuse [4]	758	2515	3283
Hate Speech and Offensive Language [5]	20620	4163	24783
Total	79558	399639	479197
Total After Removing Duplicates	**49683**	**213165**	**262848**

The total number of unique samples available is **262,848**. All of the samples are English texts, with binary labels to classify them as either offensive or inoffensive. Only **18.9%** of texts in the dataset are offensive, but we decided to keep the dataset imbalanced to simulate real world conditions where most texts would be innocuous and inoffensive.

The combined dataset was split into training, validation and testing sets in the ratio of **70:15:15**, with each split maintaining the same distribution of 18.9% offensive samples. These splits were used for qualitative testing of the system.

For quantitative testing, we used a **Wikipedia dataset of 7.8 million samples** [19].

3.2 Federated Learning

We now define a single learning iteration, which we call an **aggregation cycle**.

Let G_t be the server model at iteration t. A copy of G_t is sent to n clients, where the i^{th} client holds a unique, labeled dataset d_i, relevant to the task. Each client then proceeds to finetune (using deep learning) their copy of G_t on their dataset, and sends back the new model to the server, to create a pool of finetuned client models $C_t = \{c_{(t,0)}, c_{(t,1)}, \dots, c_{(t,n-1)}\}$. The server now uses an aggregation function f to merge all $c_{(t,i)}$ and create a new server model G_{t+1}. This process can be seen in the equation below.

$$G_{t+1} = f(C_t)$$
$$where, \quad C_t = \{c_{(t,i)}\} \qquad \forall\, i \epsilon [0, n-1]$$

In our system, the aggregation function f can be one of two algorithms, both of which are described below.

FedAvg [12]

$$f(C_t) = \frac{1}{n} \sum_{i=0}^{n-1} c_{(t,i)}$$

Calculates a simple mean of the parameters of all the models.

WMean

$$f(C_t) = \frac{1}{n}\sum_{i=0}^{n-1} \frac{|d_i|}{\sum_{j=0}^{n-1}|d_j|} c_{(t,i)}$$

Calculates a weighted mean of the parameters of all the models, where each model is weighted by the ratio of the number of samples it was trained on versus the total number of samples seen by all the client models.

Both algorithms perform equally on ***Independent and Identically Distributed (IID)***[1] data (see footnote 1), however *FedAvg* is greatly outperformed by *WMean* for non-IID data.

3.3 MapReduce Operations

We can reduce the *WMean* function into two basic operations, as shown below;

$$Let \quad w_i = \frac{|d_i|}{\sum_{j=0}^{n-1}|d_j|}$$

$$thus, \quad f(C_t) = \frac{1}{n}\sum_{i=0}^{n-1} \frac{|d_i|}{\sum_{j=0}^{n-1}|d_j|} c_{(t,i)} = \frac{1}{n}\sum_{i=0}^{n-1} w_i c_{(t,i)}$$

$$= \frac{w_0}{n} c_{(t,0)} + \frac{w_1}{n} c_{(t,1)} + \ldots + \frac{w_{n-1}}{n} c_{(t,n-1)}$$

As shown above, the product of the model parameters $c_{(t,i)}$ and the constant $\frac{w_i}{n}$ can be calculated during the **map phase**, and then the sum of the results can be computed pairwise in the **reduce phase**.

The same process can be used to compute *FedAvg*, with all w_i set to a value of 1.

3.4 Cloud Architecture

We built a federated learning system with simulated clients to test it, where a global text classification model is refined over multiple aggregation cycles to classify texts as either objectionable or not. Figure 1 showcases the architecture in further detail.

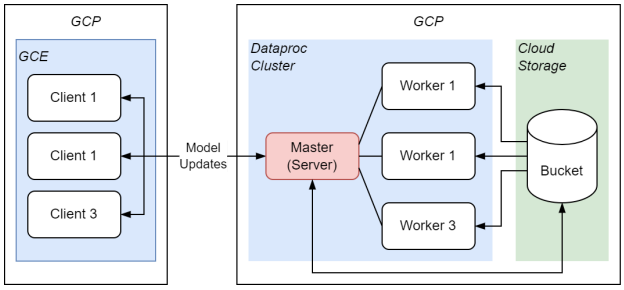

Figure 1: System architecture diagram.

As can be seen in the diagram, the system consists of a server-client architecture. Their main components are described below;

1. **Server**
   a. **Master Node:** The master node serves an API for handling server-client communications. The API sends clients the current server model when requested, and receives finetuned models back from them. The master node also communicates with the worker nodes for model aggregation by the delegation of MapReduce jobs. Additionally, the master node performs model evaluation and calculates performance scores for the aggregated models.
   b. **Worker Nodes:** The worker nodes handle the MapReduce jobs assigned to them by the master node.
   c. **Storage Bucket:** The storage bucket holds the current server model and all the corresponding client models. This bucket is accessible to all nodes on the cluster, although only the master nodes have write permission.
2. **Client**
   a. **Client Node:** The client nodes fetch copies of the server model from the master node and finetune these copies on the data they hold. Once finetuned, the client models are sent back to the server.

3.5 Infrastructure and Technical Specifications

The entire system (i.e. the server and the clients) is hosted on the **Google Cloud Platform (GCP)**. The server runs a

[1] In our experiments we distributed our training split equally between all the clients, with all of them having the same number of samples, and the same distribution of offensive and inoffensive tweets (18.9%). The samples also had no correlation to each other. This is an example of **Independent and Identically Distributed (IID)** data. In real world scenarios it is unlikely that data on any two clients would exhibit the exact same characteristics, especially when it comes to personal information like user chats, and thus the data distribution would never be IID, however, IID data serves our purpose of qualitative evaluation.

Dataproc cluster with a single master node and multiple worker nodes. The master node and worker nodes are all **Google Compute Engine (GCE)** instances of virtual machines running Linux on E2 processors with 8 GB of RAM and 50 GB of hard disk space. The master node is also equipped with a GPU to handle model evaluation. There is also a **Google Cloud Storage** bucket on the server for holding models.

The clients are identical to the master node, with all the same specifications, including GPU capability for performing deep learning tasks.

On the software side, both the server and clients run **Python** applications. The master node runs an API written using the **Flask** microframework. The cluster runs a **Spark** backend that is interfaced with using **PySpark**. We use **Pytorch** on both the server and the clients for deep learning tasks. The starting model for the federated learning setup is the transformers based **Distilbert language model** pretrained on the English language and appended with a sequence classification head, sourced from 🤗 **Hugging Face**. We use **Google Cloud APIs** to interface with GCP services.

4 Evaluations, Inferences and Results

We evaluated our model both qualitatively based on ML performance scores, and quantitatively based on performance at scale. These evaluations and their results are outlined below.

4.1 Comparing Deep Learning to Federated Learning

A study was conducted to compare a model trained using deep learning and a model trained using federated learning on the same training split (183,993 samples) with 16 clients, and the data distribution between clients being IID. These models were evaluated on the test split (39,428 samples) based on their F1-scores.

We started by evaluating the untrained model, which achieved an F1-score of **0.45** and served as our baseline. The DL model achieved a score of **0.88** in just one epoch. In contrast, the FL model was trained for up to a 100 epochs, and achieved a maximum score of **0.76**.

Figure 2 shows this comparison, and also shows the FL model improving with the number of training cycles. **FL algorithms such as *FedAvg* and *WMean* however, are generally inferior to DL optimizers such as *Stochastic Gradient Descent* and *Adam***, and it is unlikely that the FL model would have surpassed the DL even with unlimited training. We observed the FL model starting to plateau at around the 80 epoch mark proving this point, and due to time and resource constraints we decided to stop training the model at a 100 epochs.

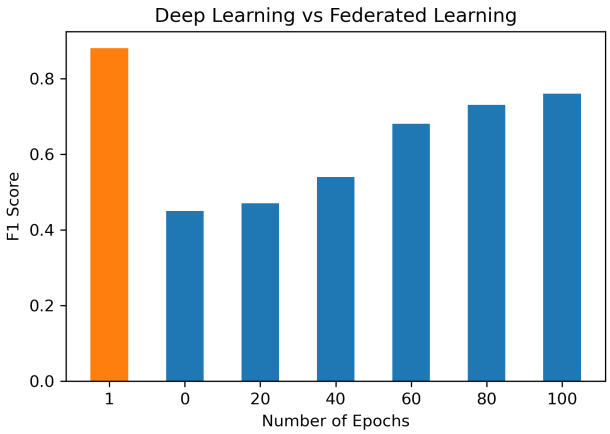

Figure 2: Comparison between deep learning and federated learning. The deep learning model is indicated by the orange bar.

4.2 Impact of Data Size on Training Duration

We evaluated the impact of data size on the amount of time it takes to train a single client model. To do so we used a large dataset consisting of 7.8 million samples sourced from Wikipedia [19]. The model was trained on progressively larger subsets of this data, while recording the corresponding training times, which can be seen in figure 3.

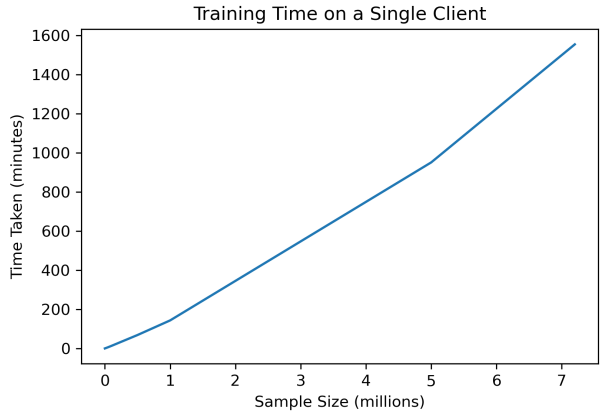

Figure 3: Model training time on increasing sample size

Our findings indicate that the training time of the model exhibits an almost linear relationship with the number of

samples used for training. Furthermore, we observed that there is always a minimum overhead of approximately 24 seconds, incurred as a result of the time taken to load the model.

4.3 Impact of Client Count on Training Duration

The number of clients is directly equivalent to the number of client models that the system needs to aggregate. To investigate the impact of clients on the training time of the model, we finetuned progressively increasing numbers of clients, from 1 to 16 using a dataset of 5 million samples from Wikipedia. The training time for each client count was recorded and can be seen in figure 3.

Figure 3: Change in training time on increasing sample size. The bar in orange indicates a regular deep learning model.

Since the total data size remains constant over the experiment, as the clients increase, each individual client has to train on a smaller share of the dataset. Additionally, all the clients train in parallel. Thus, our results show an exponential reduction in training time with an increasing client pool. Note that when there is only one client, we are essentially performing regular deep learning and not federated learning, which is indicated by the orange bar in figure 1.

This is a good demonstration of the scaling capabilities of FL, as **FL systems can work through extremely large amounts of data, as long as they have enough clients that it is split between**.

4.3 Impact of Client Count on Model Aggregation Duration

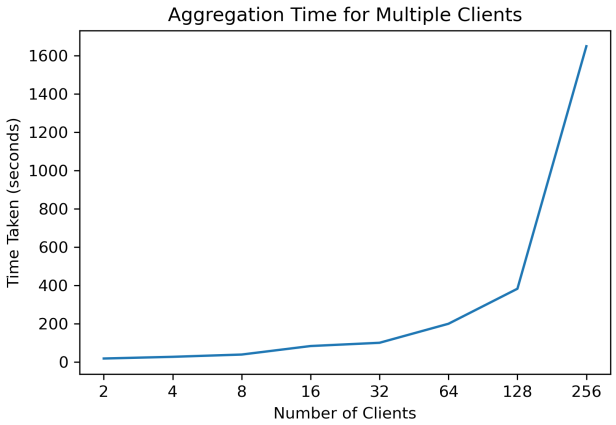

Figure 4: Model training time on increasing sample size

To analyze the impact of the number of clients on time taken for model aggregation, we ran model aggregation on progressively larger pools of client models. The aggregation was performed using a fixed set of 4 workers.

We see that **aggregation time rises along with the number of clients**. There is also a sharp increase at the 128 clients mark, which can be explained by the requirement for a lot of data shuffling in memory as the models start to far outnumber the workers.

4.4 Impact of Worker Count on Model Aggregation Duration

To assess the impact of the number of workers on the time taken for model aggregation, we held the client count constant at 32 clients, and recorded the time taken to perform the aggregation by progressively increasing the number of workers from 2 to 7. The results can be seen in figure 5.

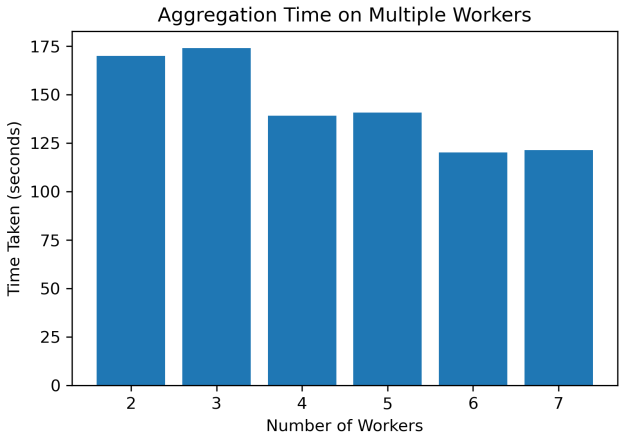

Figure 5: Model training time on increasing sample size

Our results indicate a **reduction in aggregation time as the number of workers increases**. We also observed that increasing an even number of workers by one, generally makes an insignificant amount of difference to the aggregation time. At the moment we don't have an explanation for why such a trend would occur.

4.5 Latency and Throughput

The throughput of our system was evaluated by measuring the average rate at which 5 million samples were processed. Using 16 clients and 4 workers, we achieved an average throughput of **1360 samples per second**.

In addition, we assessed the latency of our system during the inference process. Our measurements indicate an average communication time of **28.215 milliseconds** between the client and server.

4.5 Runtime Breakdown

To investigate the breakdown of runtime tasks, we measure time spent on individual tasks for a single aggregation cycle, performed over the training split with 7 workers and 16 clients running in parallel. Our analysis reveals that a majority of time is consumed during finetuning (accounting for 48% of the total time), followed by the time required for model transmission and reception, with each task consuming 24% of the total time.

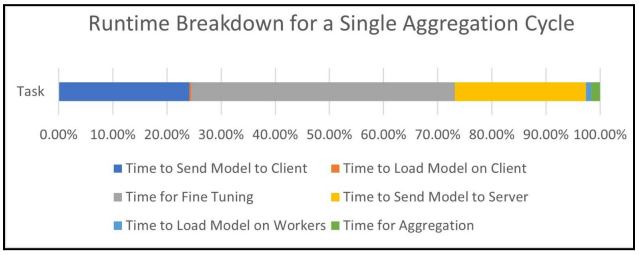

Figure 6: Breakdown of time consumed in subtasks.

Our analysis reveals that a majority of time is consumed during finetuning (accounting for **48%** of the total time), followed by the time required for model transmission and reception (each consuming **24%** of the total time).

5 Bottlenecks

We identified three main bottlenecks in our system. The most impactful one is the **number of workers**. As the number of clients rises, the server has to aggregate a greater number of models, and the model aggregation step is heavily reliant on the number of worker nodes to be able to give quick results.

The second big bottleneck is **storage and memory I/O**. The server cluster has to pull and push a large number of models from and to the storage bucket, and have to shuffle models in and out of memory during the reduce phase. This causes a significant amount of time to be spent making models available for processing.

Finally the third and final choke point is the time spent **transmitting models between the server and client**. This can mostly be attributed to network speed and bandwidth of the server and clients, and poor internet will lead to generally slower aggregation cycles.

6 Challenges

We faced a number of challenges while designing, developing and testing our pipeline. The first major challenge was **limited compute and resources**. A lot of our ML code was initially built and tested on Google Colab so that we could run our models on GPUs/TPUs, which were allocated based on availability and usage quota. We later migrated to GCP to test our web APIs, where we were only able to deploy a maximum of 8 GCE instances at a time. Thus we had to run multiple GCP accounts to be able to simulate both the server and the clients.

A second big hurdle was in the design of the server and client applications. **Federated learning has a lot of edge cases** where a client may fall out of synch with the server (client fails, model takes too long to finetune, client join in the middle of an aggregation cycle, etc) and we had to account for and find ways to deal with all of these possibilities.

During the actual running of the system, we also saw **issues due to model size**. Since deep learning models can get pretty big, we would risk running out of storage space in our bucket, or run out of memory on the instances. Larger models would also be slower to transfer between the server and the clients. To deal with this we used the smallest language model we could find without compromising too much on qualitative performance. This was the Distilbert model, which is 235.7 MB in size.

7 Conclusion

In this report we present our work on building a federated learning system in the cloud, that performs distributed machine learning on edge devices. Our system compares well to regular deep learning, with the added advantage of preserving data privacy, and scaling to millions of samples in a fraction of the time taken by the deep learning models.

Our evaluations showcase many pros and cons of FL, and our key learnings are listed below;

1. **Pros**
    a. **High throughput:** FL can handle a lot more data and faster.
    b. **Preserves data privacy:** By sending the model to the client, they no longer have to share personal data with the server while allowing the model to learn from it.
2. **Cons**
    a. **High infrastructure cost:** Scaling FL systems relies directly on the number of workers and the amount of storage available. Systems may also need high end equipment such as GPUs/TPUs and SSDs for additional speed up.
    b. **High latency:** Depending on the size of the model, transmitting models between the server and clients can lead to a lot of overhead.
    c. **Inferior qualitative performance:** FL algorithms underperform when compared to regular deep learning optimizers and may not lead to comparative performance every time.

8 Future Work

While we have described an extensive range of evaluations in this report, there are some things we were unable to accomplish during the course of this semester. Firstly, we skipped testing our system on non-IID data. As explained in footnote 1, one would seldom deal with IID data in any real-life scenarios, and thus testing on non-IID data distribution between clients is something that we would like to experiment with and evaluate. Using non-IID data would also allow us to compare *FedAvg* to the *WMean* algorithm, of which the evaluation of the latter too was skipped this time, even though the code for *WMean* was written and is present in the final submission.

We also plan to parallelize model evaluation into a MapReduce routine, where the evaluation split can be further partitioned and each worker can load the model and its specific partition to produce predictions. These predictions can then be pooled back to calculate results. This would cause the system to speed up even further.

REFERENCES

[1] United Nations, *Understanding Hate Speech?* (link)
[2] Shane Cooke, *Labelled Hate Speech Dataset* (link)
[3] Saurabh Shahane, *Cyberbullying Dataset* (link)
[4] Curry et al., *ConvAbuse: Data, Analysis, and Benchmarks for Nuanced Abuse Detection in Conversational AI* (link)
[5] Davidson et al., *Automated Hate Speech Detection and the Problem of Offensive Language* (link)
[6] Andrii Samoshyn, *Hate Speech and Offensive Language Dataset* (link)
[7] UNICEF, *Cyberbullying: What is it and how to stop it* (link)
[8] Facebook, *Sharing Our Actions on Stopping Hate* (link)
[9] Twitter, *Hateful Conduct* (link)
[10] Reddit, *Content Policy* (link)
[11] Instagram, *How Instagram uses artificial intelligence to moderate content* (link)
[12] McMahan et al., *Communication-Efficient Learning of Deep Networks from Decentralized Data* (link)
[13] Hard et al., *Federated Learning for Mobile Keyboard Prediction* (link)
[14] Yuan et al., *A Federated Learning Framework for Healthcare IoT Devices* (link)
[15] Li et al., *A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection* (link)
[16] OpenMined, *Federated Learning for Credit Scoring* (link)
[17] Google, *Towards Federated Learning at Scale: System Design* (link)
[18] Microsoft Research, *FLUTE: A Scalable, Extensible Framework for High-Performance Federated Learning Simulations* (link)
[19] Wikipedia Dataset (link)

APPENDIX

"On my honor, as a University of Colorado Boulder student
I have neither given nor received unauthorized assistance."

I. Contributions

1. **Aditya Srivastava:** ML development, API design, server side API development, authoring Spark MapReduce routines, GCP devOps and deployment, running evaluations, authoring project documentation.
2. **Harsh Gupta:** API design, Client side application development, authoring Spark MapReduce routines, GCP devOps and deployment, running evaluations, authoring project documentation.

Most of the tasks were performed together with both partners either brainstorming together or helping each other bug fix, so even if there are differences in the work distribution, neither of the authors handled any task completely independently.