FLAT-C: An approach for Text-Classification in a P2P Network using Federated Learning

Arjun Ramesh Rao
arra8056@colorado.edu

Aditi Prakash
adpr5166@colorado.edu

Shree Krishna Subburaj
shsu1807@colorado.edu

Anusha Gupta
angu5645@colorado.edu

ABSTRACT

Encrypted communications have become increasingly common with the advent of chat services that support end-to-end encryption. Particularly in peer-to-peer chat systems like Tox [7] messages are encrypted using secure perfect forward secrecy. In such a scenario using machine learning for tasks like spam classification become increasingly challenging as text messages cannot be sent in plain text and there is no central server that collects data to train models. Recent advances in Federated Learning algorithms have provided ways to train machine learning models on the edge, without having to transport training data to a server. In this project, we compare an approach for Federated Learning in a Peer-to-Peer architecture without a central server against a Centralized Federated Learning Approach [3]. We overcome the difficulties of working with encrypted text, by using on-device machine learning and communication of local model updates to peers or the central the server, without breaching privacy or requiring transport of decrypted messages across the network.

Author Keywords

Peer-to-Peer messaging, End-to-end encryption, Federated Learning

INTRODUCTION

Unstructured text data is available in abundance due to chat applications like Whatsapp, Messenger, Telegram etc. With privacy and security becoming primary concerns for users, these applications are using encryption techniques to preserve the privacy of their users [8]. Applications like Whatsapp and Telegram apply techniques like Curve25519 based algorithms [1] and MTProto protocol [4] to ensure end-to-end encryption between the sender and recipient of the chat. Performing text classification tasks on such data becomes a challenge. To run a Machine Learning model on this data, we would need to pool data from different devices to a central server. However, this becomes infeasible not only because of the overhead to be borne by the central server in terms of data storage but also due to the privacy-sensitive nature of the data. However, in Federated Learning the central server computes a global model of weights based only the local weight updates received from the clients in the network as opposed to the actual data. Such an approach can overcome the challenges of not having to send raw unencrypted text or requiring the server to have capabilities to decrypt messages and requires less storage on the server side.

Similarly, in a P2P setting, when using machine learning for spam detection, we can overcome the problem of encrypted text by training on the text locally in each node where it can be safely decrypted and only transfer the learned weights and parameters to peers. Instead of requiring that all peers send weight updates to every other peer in the network, the learning can be restricted to a specific number of peers based on certain strategies like n-nearest peers, n-random peers etc or a ring topology based system where updates are done in a circular fashion among neighbors in a ring.

Both these methods overcome the issues discussed previously regarding the overhead and the sensitive nature of the data. In the following sections, we describe the related literature, and our proposed approach to Federated Learning in both p2p and centralized settings. We discuss the experimental setup, results, conclusions and provide a note on future work.

RELATED WORK

Federated Learning is a form of collaborative learning that shares a prediction model across the different nodes. Federated Learning algorithms may use a central server that orchestrates the different steps of the algorithm and acts as a coordinator, or they may be implemented in a peer-to-peer fashion where the nodes in the network directly communicate with each other. Centralized server approaches have been successfully used to learn a shared model by aggregating updates from models on distributed devices [3]. Recently peer-to-peer approaches have also gained traction. One approach involved judicially aggregating information from local observational data of one-hop neighbors, and thus collectively train a model that best fits the observations over the entire network [2].

Another recent approach for peer-to-peer Federated Learning on medical imaging data [5], used a randomly selected client that sends requests to other peers for their local model updates. These updates are averaged by the selected client and then

used by the client as its starting model for the next round of updates. Similarly in each round different peers are chosen to receive model updates and the training proceeds in this manner. In our project, we plan to experiment with different strategies for sharing model updates among peers and compare the peer-to-peer approach with the centralized one.

EXPERIMENT

Dataset

We utilized UCI's SMS Spam Collection Dataset [6] consisting of 5574 instances for the spam classification task. We used a 80-20 train-test split, and use the test set as a common benchmark for evaluating our models. The training set is split among the clients in different experiments primarily based on 2 different strategies.

Dataset 1: Splitting based on number of clients:
In this strategy we divide the entire training data into N splits, one for each of the N clients in the network. Here the entire training data is available to the pool of clients and no data is left out in any experiment when using this strategy. So here increasing the number of clients does not necessarily provide more information in the overall network as even a network with just 2 clients has all the data available among the two clients as does a network with say 5 clients.

Dataset 2: Using a fixed N-partition split:
In this strategy we first split the training data into N partitions, and then depending on the actual number of clients in a network M where $M <= N$ we only use the first M partitions from the training data. This approach simulates a more realistic scenario where clients in a network will not have all the data available among them, and increasing the number of clients provides more data in the network overall to improve the Federated Learning model.

Federated Learning Algorithm

There have been previous attempts to train neural networks using a Federated Learning algorithm in a peer-to-peer fashion. One such approach involved updating a single random client at every round and using a weighted averaging technique to update the client's weights [5]. This is in contrast with using a central Federated Learning Server where all clients are updated at each round. For our experiments, we follow a of combination of both approaches, where all clients are updated every round in a peer-to-peer fashion but each peer only receives updates from a subset of nodes in the network.

We compare the peer to peer architecture with a centralized Federated Learning approach. Algorithm 1 describes the local weight update algorithm that we use for our experiments which is essentially the popular gradient descent algorithm used in common Machine Learning problems. Algorithm 2 describes how Federated Learning works among different clients. In case of the Centralized Server approach, Algorithm 2 runs on the server, while Algorithm 1 runs on each client. In the peer to peer case, both Algorithm 3 runs on each client.

The key point of difference between Algorithm 2 and 3 is that at the start of each round, clients do not start with the same weight in case of Algorithm 3, while in Algorithm 2 in the beginning of each round, all clients have common weights.

Algorithm 1 *LocalUpdate*

for each local epoch **do**
  for each batch b **do**
    $w = w - \eta \cdot \nabla l(W;b)$
  end for
end for

Algorithm 2 *AggregateModel*

Initialize W_0
for round r in $1,2\ldots,m$ **do**
  for client i in $1,2\ldots,N$ **do**
    $W^i_{r+1} = LocalUpdate(i, W_r)$
  end for
  $W_{r+1} = \sum_{k=1}^{N} \frac{n_k}{n} W^k_{r+1}$
end for

Algorithm 3 Peer2Peer Update

Initialize W_0
for round r in $1,2\ldots,m$ **do**
  for client i in $1,2\ldots,N$ **do**
    W^i_{r+1} = LocalUpdate(i, W^i_{r+1})
    W^k_{r+1} = GetWeightsFromRandomPeer()
    $W^i_{r+1} = \frac{(W^k_{r+1} + W^i_{r+1})}{2}$
  end for
end for

In Algorithm 3 the function GetWeightsFromRandomPeer uses different strategies depending upon the experiment being considered.

Design

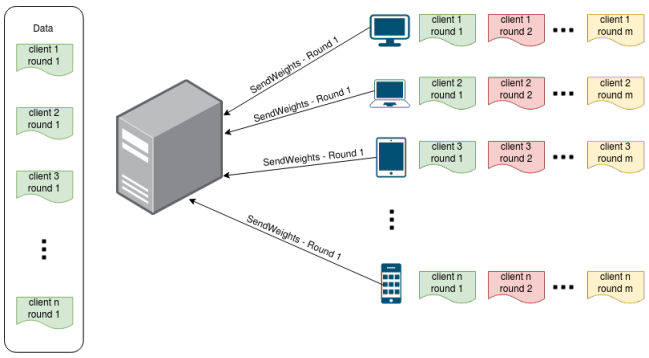


Figure 1: Weight aggregation

All our experiments involve 5 rounds of Federated Learning. We experimented with various round numbers and fixed on having 5 rounds to have sufficient training data in each round as well as have enough rounds to be able to see its effect of

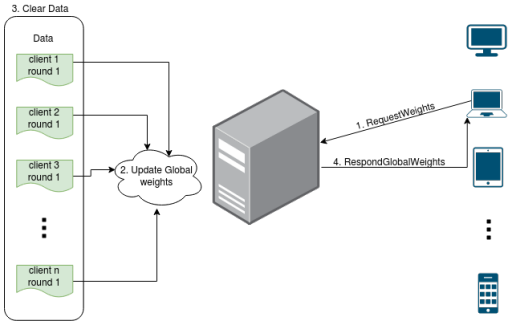


Figure 2: Global model request and return

on performance. We define a round as one set of operations involving a local model being trained on each client, weight updates exchanged among clients, and a new model trained using the received weight updates. A round ends when a client has computed an aggregate model (either by using weights from the central server or by averaging weights from peers).

Federated Learning using a Central Server:
In this experiment, all clients in the network have some arbitrary amount of text data which is split into parts for each of the 5 rounds of Federated Learning. The server maintains a buffer where it stores the data received from the clients of the network. In round 1, the data buffer of the server is initially empty and each client trains the model on their local data using the same set of initialization weights received from the server. At the end of each round, the clients send their respective learned weights to the server which gets stored in the server's data buffer as shown in Figure 1. Once all the updates have been received, the server then computes a global model by proportionally averaging weights received from the clients based on their local dataset size. Once the new global weights are computed, the server responds to the client with the global weights and clears its own data buffer as shown in Figure 2. Since the global weights and the local weights from each client in the server are shared resources, we use a mutex lock to ensure only one client at a time accesses these resources.

We conducted multiple experiments using the central server, by changing the number of clients in the network, as well as using the two different data splitting strategies as mentioned in the Dataset section.

Federated Learning using a P2P network:
We setup two kinds of networks for the experiments in the P2p setting. One is a mesh network where each peer can communicate with any other peer in the network, and the other is a network with a logical ring topology, where a peer can only communicate with the next peer in the ring. We varied the number of peers in the network using both data splitting strategies similar to the experiments conducted with a central server. We also conducted experiments by varying the number of peers that are contacted for weight updates in a round. In these set of experiments we start by having each peer request updates from one random peer in the mesh network, and increase the number of random peers from 1 to $N-1$ where N is the number of peers in the network. Additionally, we use the ring topology in one experiment where each peer only requests for weights from the immediate peer in the ring.

Evaluation Methods

We plan to use two types of evaluation metrics that compare the models that are trained by following the traditional machine learning approach and the peer-to-peer Federated Learning approach. The metrics are listed as follows:

- **Machine Learning Metrics:** F1 Score, AUROC score.
- **Network Performance metrics:** Number of messages sent/received (Message Volume).

RESULTS

Based on the aforementioned experiments, the following tables and plots summarize our results. Tables 1 and 2 show the average F1 scores at the end of round 5, along with the total Message Volume at the end of round 5 for both Centralized and P2P experiments using the two different Dataset splitting approaches mentioned earlier. The performance when using different weight update strategies in a fixed size (5 peers) P2P network is shown in Table 3.

Table 1: Average F-1 score and Message volume at Round 5 (using Dataset 1)

Clients	Centralized		P2P	
	Avg F1	Msg Volume	Avg F1	Msg Volume
2	0.85	390	0.85	330
3	0.80	585	0.80	990
4	0.73	780	0.71	1980
5	0.75	975	0.72	3300

Table 2: Average F-1 score and Message volume at Round 5 (using Dataset 2)

Clients	Centralized		P2P	
	Avg F1	Msg Volume	Avg F1	Msg Volume
2	0.7	390	0.69	330
3	0.73	585	0.71	990
4	0.73	780	0.71	1980
5	0.75	975	0.72	3300

Table 3: Average F-1 score and Message Volume at the end of Round 5 for various weight exchange methods in P2P networks having 5 peers.

Weight updates from	Avg F1	Message Volume
1 random peer	0.75	819
2 random peers	0.80	1638
3 random peers	0.73	2457
4 random peers	0.75	3300
random random peers	0.74	2244
Neighbors in ring	0.74	735

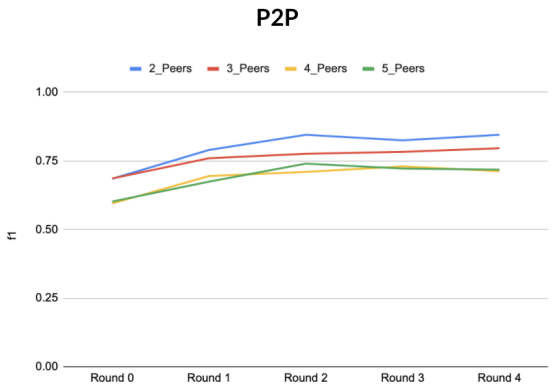


Figure 3: Average F1 Scores vs Rounds for different P2P network sizes using Dataset 1.

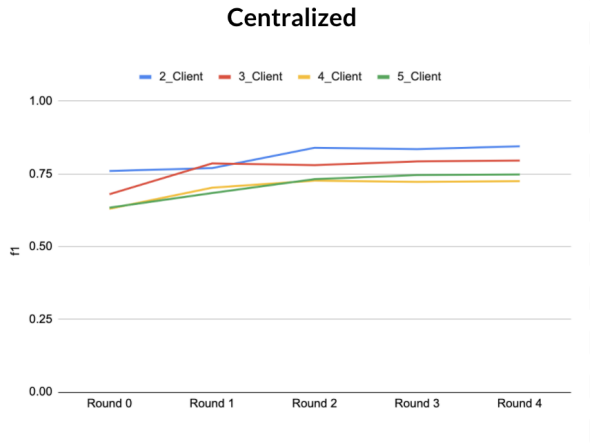


Figure 4: Average F1 Scores vs Rounds for different Centralized network sizes using Dataset 1.

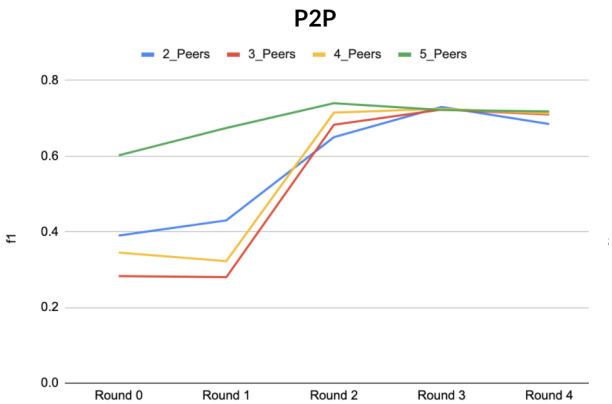


Figure 5: Average F1 Scores vs Rounds for different P2P network sizes using Dataset 2.

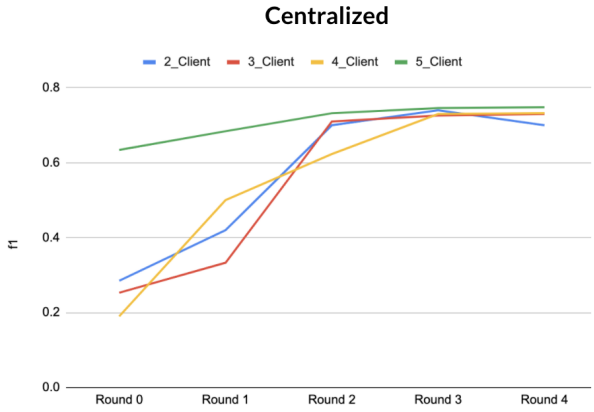


Figure 6: Average F1 Scores vs Rounds for different Centralized network sizes. Dataset 2.

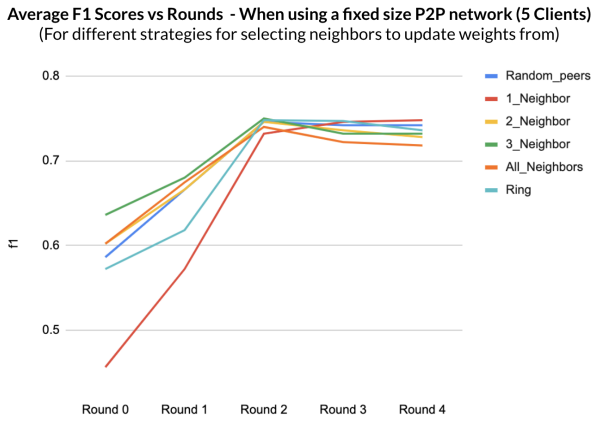
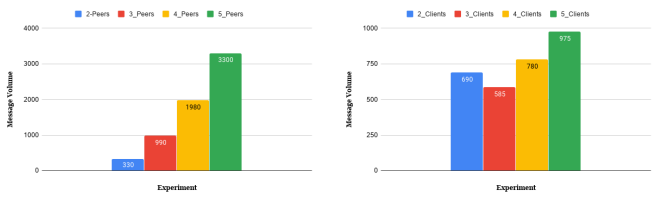


Figure 7: Average F1 Scores vs Rounds - When using a fixed size P2P network (5 Clients)



(a) P2P (b) Centralized

Figure 8: Message Volume in P2P vs Centralized for different Network sizes

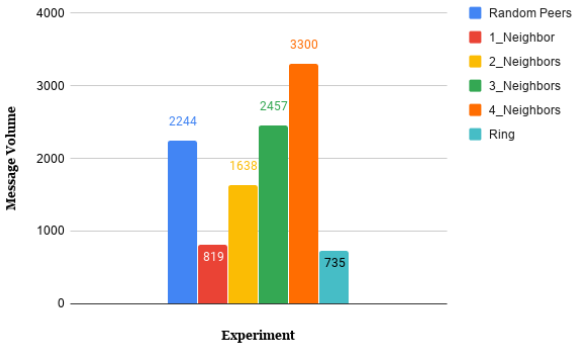


Figure 9: Message Volume for different P2P experiments (Fixed size network - 5 clients, for different neighbors chosen for weight update)

The variations in F1 scores between rounds of Federated Learning in Centralized and P2p networks of different sizes when using the 2 different dataset splitting approaches are shown in Figures 3, 4, 5, 6. Figure 7 shows the variations between rounds when using different P2p weight update strategies for a fixed size network with 5 peers. The message volumes in different network sizes for P2p and centralized approaches are shown in Figure 8, and the message volume for different P2P update strategies is shown in Figure 9.

CONCLUSIONS

Based on our experiments in different centralized and P2P settings, we found the performance of the models trained using both approaches were comparable, with a marginal improvement in performance in the centralized case. While increasing the network size incrementally, we observed that the P2P approach resulted in a substantial increases in the message volume. Whereas for the centralized approach, the increase in message volume was not significant. However, the single point of failure is one downside to the centralized approach. It is seen that the F1 score peaked at the end of the round 2 in both centralized and P2P setting, and additional rounds only resulted in marginal improvement. In order to discern the effect of Federated Learning, we experimented with two data splitting strategies. By holding more data initially, a node is able to train an accurate model in just one round, and therefore produces a smooth increase in performance. This is indicative of the pronounced effect that availability of training data has on the performance. Even when each node has less amount of data, we found that the performance peaked at the end of round 2, clearly illustrating the benefits of Federated Learning. Finally, by using a ring topology or receiving weights from just 1 random peer, we were able to achieve similar performance to other approaches where the message volume was significantly higher. This leads us to believe that it is possible to further optimize performance by using different peer topologies and strategies to exchange weights.

FUTURE WORK

In our analysis, we consider the F1 score and message volume metrics to gauge the performance of models. For a holistic evaluation of a model, our analysis can be extended by utilizing more elaborate metrics that describe network traffic, time to train, bandwidth usage, cpu usage, disk utilization etc. Also, based on our convincing performance using ring topology, there is sufficient motivation to explore additional topologies such as bus, tree and star in the P2P setting. Finally, in order to evaluate the practical utility for real-time applications, it is meaningful to test the generalizability of these approaches on more sophisticated ML techniques like CNNs, RNNs, etc for complex machine learning problems like Image Classification, Object detection, etc.

REFERENCES

[1] Ashish Bijawat. 2016. How WhatsApp Uses End to End Encryption? (2016). `https://www.linkedin.com/pulse/how-whatsapp-uses-end-encryption-ashish-bijawat`.

[2] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. 2019. Peer-to-peer Federated Learning on Graphs. (2019).

[3] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and others. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[4] MTProto 2017. MTProto Mobile Protocol. (2017). `https://core.telegram.org/mtproto`.

[5] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. 2019. BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning. *CoRR* abs/1905.06731 (2019). `http://arxiv.org/abs/1905.06731`

[6] José María Gómez Hidalgo Tiago A. Almeida. 2011. SMS Spam Collection v.1. (2011). `http://www.dt.fee.unicamp.br/~tiago/smsspamcollection`.

[7] tox 2014. Tox: a peer-to-peer protocol that offers end-to-end encryption. (2014). `https://toktok.ltd/spec.html#introduction`.

[8] Kurt Wagner. 2015. Is Your Messaging App Encrypted? (2015). `https://www.vox.com/2015/12/21/11621610/is-your-messaging-app-encrypted`.