Collaborative Cognition: An Implementation of Federated Learning

Alex Racapé
Bowdoin College
aracape@bowdoin.edu

Zeb Becker
Bowdoin College
zbecker2@bowdoin.edu

I. Introduction

In March, Italy's data protection authority banned ChatGPT due to privacy concerns.[1] The model was trained on millions of web pages, books, and Reddit conversations, and in this mix, the model was fed personal information for some internet users. With this incident, Italy was the first government to ban a chatbot, claiming that this data was illegally collected from users without an age-verification system to protect minors. While Italy lifted the ban in late April, other countries are looking into investigating the company's data practices, underscoring a fundamental clash between innovation and privacy. Data is at the core of machine learning, yet traditional methods of training rely on aggregating enormous amounts of data regardless of privacy concerns.

Federated learning is a paradigm where data is protected by introducing a decentralized framework that redistributes the learning process across multiple devices or servers. Federated learning capitalizes on the nature of distributed data by allowing individual devices, such as smartphones, to perform local computations on their respective data sets. These local models then work collaboratively to aggregate changes into a new global model, reflecting the collective knowledge gathered from the decentralized training process. The essence of federated learning lies in its ability to harness insights from diverse data sources without necessitating the transfer of raw data to a centralized location. Using techniques such as differential privacy and secure aggregation, federated learning systems can also guard against revealing information about sensitive data in the model updates that are sent over the network. However, this is beyond the scope of our project.

This decentralized approach not only safeguards user privacy, but it also aligns with evolving regulatory frameworks that prioritize data protection. For example, in healthcare, many hospitals are not allowed to release patients' personal data which results in an obstacle for applying ML models to medical tasks.[2] Computer vision models could be used to help with imaging data, and federated learning could unlock vast sources of data that are currently underutilized. Distributing data can also help companies avoid data breaches and costly regulatory penalties. For example, Uber had to pay $148 million in settlements after breaching personal information for 600,000 drivers in 2016.[3] Federated learning can be applied to other contexts where it is impractical or infeasible to aggregate large datasets centrally. Edge devices, such as smartphones, IoT devices, and environmental monitors, possess unique data that can be leveraged to enhance models through coordination.[4] Federated learning facilitates the use of this diverse data without the need for a cumbersome, centralized repository, making it a versatile solution for applications ranging from personalized recommendation systems to

[1] Adam Satariano, "Chatgpt Is Banned in Italy over Privacy Concerns," The New York Times, March 31, 2023, https://www.nytimes.com/2023/03/31/technology/chatgpt-italy-ban.html.

[2] Mohammad Tajabadi et al., "Sharing Data with Shared Benefits: Artificial Intelligence Perspective," Journal of Medical Internet Research, March 23, 2023, https://www.jmir.org/2023/1/e47540.

[3] Qinbin Li et al., "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection," December 5, 2021, https://arxiv.org/pdf/1907.09693.pdf.

[4] Li Li et al.,"A review of applications in federated learning," Computers & Industrial Engineering. Volume 149, 2020. https://doi.org/10.1016/j.cie.2020.106854.

healthcare diagnostics.

II. Related Work

In the literature, several papers have explored various approaches to federated learning. Existing approaches in industry from companies like Apple and Google mainly focus on mobile devices, but their designs share some similarities with our system–most notably, the central coordinating server model. Workers send updates to a central server, and an aggregation function incorporates these updates into a new version of the model. Apple utilizes federated learning for automatic speech recognition for products like Siri.[5] Their approach is similar to our system as it attempts to abstract the specific computations performed on the worker device into a separate plugin. Rather than implementing task specific code as a core part of the system, their work focuses on providing a generalizable framework for tasks.

Google has used federated learning for large scale systems with a focus on synchronous systems.[6] While there have been some studies using asynchronous training, most work in production uses synchronous rounds of training and the federated averaging function to aggregate updates from workers. FedAvg is essentially a simple weighted sum across the updates submitted by the workers, where each set of weights is weighted by the amount of data the worker is trained on. In other words, contributions from workers with more data should be weighted more heavily than updates from workers with relatively little data. In production, they have used this architecture in the context of phone keyboards and text suggestions. Their work focuses on addressing issues such as varying availability and differences in compute resources as they have scaled their "lock-step execution" to tens of millions of devices.[7]

III. Design and Architecture

Figure 1 depicts the structure of our implementation, which revolves around two primary entities:

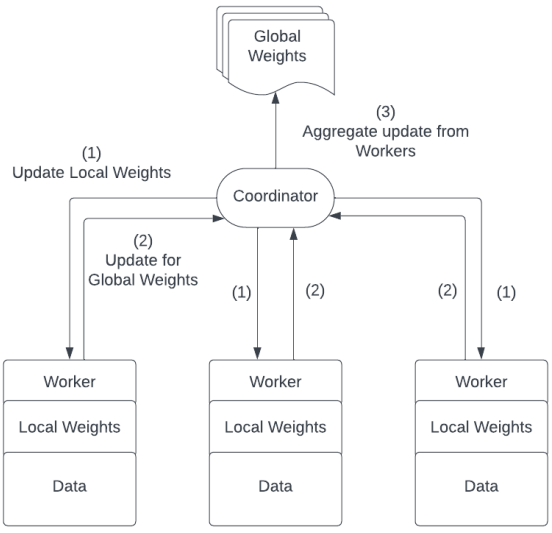

Fig. 1. General Structure

workers and a coordinator. Workers each have a unique dataset and train locally for a period of time after requesting weights from the coordinator. When training locally, each worker uses stochastic gradient descent for a number of epochs. The coordinator is started in a process completely independently from the other workers. The coordinator has no knowledge of the other workers ahead of time, and they can join the training process at any point. The coordinator's main role is to manage the state of the global weights. To do this, it must keep track of some information about each worker as it joins the network in order to merge the updates from the workers for each round of training. Relatedly, the coordinator needs to synchronize workers so that training proceeds in organized rounds. This is necessary because each worker must start from the same version of the weights for their updates to be aggregated later. In federated learning there are several options for aggregator functions, but our implementation uses federated averaging since this is one of the simplest and most prevalent functions. Previous empirical studies have also shown that it performs well for both homogenous and heterogenous data settings.[8]

[5]Matthias Paulik et al.,"Federated Evaluation and Tuning for On-Device Personalization: System Design & Applications," February 16, 2021. https://arxiv.org/pdf/2102.08503.pdf.

[6]Keith Bonawitz et al.,"Towards Federated Learning at Scale: System Design," March 22, 2019. https://arxiv.org/pdf/1902.01046.pdf

[7]*Ibid.*

[8]Liam Collins et al., "FedAvg with Fine Tuning: Local Updates Lead to Representation Learning," May 27, 2022. https://arxiv.org/abs/2205.13692

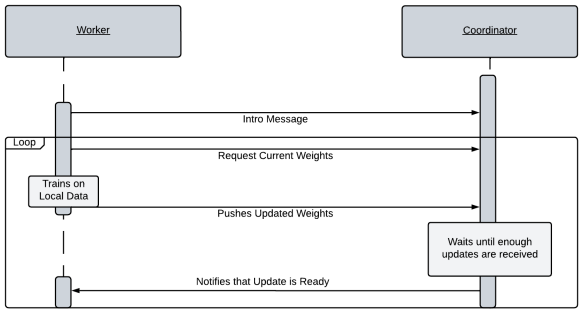

Fig. 2. Worker-Coordinator Communication Flow

In our system, the coordinator must receive enough updates from workers to move on to the next round of training, exceeding a certain threshold set by the user. For some applications, it may make sense for there to be little tolerance for failure. Therefore, this fraction can be set to one, indicating that all workers must reply in order for training to proceed. However, it may make more sense to move on to the next round of training without responses from some workers in some applications with many edge devices. One such context could be training on mobile devices which are much less reliable. When we have enough responses, we call this a quorum. There are two added benefits from this quorum protocol. First, the coordinator will not stall if workers fail or leave the network unexpectedly. Second, this process addresses stragglers, and the training process can continue with less downtime. However, this means that the coordinator must add a mechanism for load balancing or else the same machines will consistently fail to contribute to updates. For this, our coordinator uses a simple load balancing process that is inspired by TCP and AIMD. If a worker responds quickly, the coordinator will increase that worker's workload by incrementing the number of epochs it will run locally for each update. If a worker is late to respond, the coordinator will divide the worker's load in half by reducing the number of epochs it is assigned.

When the coordinator obtains enough responses from the workers, the coordinator aggregates the new weights and sends an update to notify workers that an updated model is ready. This loop of communication is illustrated in Figure 2. We settled on this notification approach after considering two other alternatives. We could have had the coordinator send the updated global weights when responding to the worker's RPC call that sent its local findings. With this approach, we would have to keep a large number of sockets open at once while waiting for worker responses, resulting in a bottleneck at the coordinator. Alternatively, we could have workers send their updates then poll periodically to check for a new round of training. However, this would result in a large amount of network traffic, and the coordinator would still have to service all of these polling requests. With the notification approach, the coordinator sends a single message to workers, and each worker then requests the newest version of the model. It is important to note that there is some added overhead for each worker which must run an RPC server in a background thread to receive these notifications. As an added benefit, the coordinator can look for dead workers by checking if a worker ever responded to its notification. Once a worker receives a notification, it requests the newest model weights, and if a worker is still training, it will see the notification and quit training to start over with the new weights. This completes the communication loop in Figure 2, and the cycle will continue until the coordinator terminates the process after a specified number of training rounds.

IV. Implementation

To build our architecture we used Python, XML-RPC, and PyTorch. We chose Python so that we could integrate with popular machine learning frameworks like Pytorch. Our implementation is inspired by MapReduce with the goal of making the system as modular and customizable as possible. All of the machine learning code is distinct from the communication and coordination code. Workers can easily specify a different data source, and code for the model specification is separated into a separate configuration file. With our system, users can specify their model, and our code abstracts away the details of coordinating across workers. While our workers all use Pytorch, messages between the coordinator and workers are language and framework agnostic. This allows peers to theoretically use other implementations in the future to collaborate across different languages.

In terms of our testing application, we chose to simplify the machine learning code by working with the MNIST digits dataset. Our goal with this system was to focus more on the framework for training. With our system, users can customize to work with nearly any model or dataset. By simplifying the learning problem, we focus more on the system's design. Additionally, our testing machines did not have GPUs, so it would not be worth slowing down the training process with an overly complex problem. We also added logic so that each machine worked with a random subset of the MNIST dataset. When randomizing this set, we also varied the number of total images for each worker. To facilitate testing on a large number of AWS EC2 machines, we developed several scripts to deploy and run code across all of the machines.

Although our system is flexible and easy to join, it is susceptible to several security threats. In a model poisoning attack, a malicious entity could join the network as a worker to manipulate the weights of the model. This worker could even claim to have enormous amounts of data, so that its harmful update is weighted even more by the coordinator. Additionally, an attacker could flood the coordinator with requests to join the network which would ultimately terminate training. For our implementation, we did not focus on security concerns, but this is one area that could be improved in the future.

V. Evaluation

We tested our system using a set of 50 commodity servers spread across multiple locations around the world. We deploy the worker program and model configuration to each machine. To execute an experiment, we configure the coordinator and worker settings as specified for each experiment, start the coordinator server on one machine, and then start the worker program on each other machine, passing in the IP address of the coordinator server and the public IP address that the worker should use for its XMLRPC server.

A. Accuracy

We determined the accuracy of a trained model by testing it against a subset of data reserved for testing. The accuracy metric in our experimental task is the ratio of test images that the model correctly classified to the total number of test images. Ideally, the accuracy of a federated model trained on a given dataset split across many machines will be similar to the accuracy of a traditional model trained with the complete dataset on a single machine. It would be concerning if, when we split a dataset across two machines, the accuracy dropped by half.

Fig. 3. Test Accuracy of N Workers

To test the performance of our system, we defined a subset of 40,000 MNIST images as our training dataset. We trained for two global epochs, so that each worker would get averaged weights that took into account updates from each other worker in the second epoch. Each worker tested their model on a subset of their local data, then sent their accuracy metric to the coordinator, which combined them into a mean accuracy. The mean accuracy for the second epoch was recorded as our accuracy metric.

For each round of testing, we split the same dataset into smaller, non-overlapping subsets that were distributed to each machine. The first round of testing evaluated one worker with 40,000 images, the second round evaluated two workers with the same dataset split to give 20,000 images to each, and so on. In order to ensure that every worker had the chance to contribute its update to each global update, we set the quorum percentage to 100%, essentially turning off the quorum mechanism and requiring that every worker respond before the coordinator would initiate a new global epoch. The accuracy of each data set for N workers is shown in Figure 3.

In Figure 3, we saw a slight decrease in accuracy as we split the dataset over more workers, but no dramatic change. One worker achieved an accuracy of 97.76%, and eight workers achieved an accuracy

of 95.02%. In general, as the number of workers scaled exponentially, the accuracy decreased only slightly, showing that our design is scalable and can achieve comparable results to a traditional system. The limitations of our experiment are twofold. First, because our model uses stochastic gradient descent, it is impossible to exactly compare results across model training runs. Rather, experiments like this serve as a sanity check ensuring that system performance remains within an acceptable range. Second, the procedure used in this experiment is not feasible for evaluating scalability with real world data. To compare 1-worker runs with 8-worker runs, for example, we will at some point need all of the test data on a single worker. However, doing this with real data would completely defeat the initial motivation for federated learning- keeping data private and decentralized.

B. Load Balancing

In order to test our load balancing system, we recorded the round trip training time for each worker at each epoch. The round trip training time is defined as the total time between when the coordinator sends a set of weights to a worker and when the coordinator receives that worker's updated weights back.This value is measured at the coordinator. Another way to measure training load would be to time the completion of one global training round at the worker. This would capture differences in computational power available to each worker. However, by measuring at the coordinator, we ensure that we are also accounting for network latency. Because the coordinator waits for most of the workers to respond before starting a new epoch, the limiting factor is round trip time, not just computational time, for each worker. Therefore, measuring at the coordinator is more desirable. To test our load balancing mechanism, we trained for ten global training rounds with a quorum percentage of .8, recording the roundtrip times for each worker at every epoch as shown in Figure 4.

Figure 4 shows mixed effectiveness. Most workers are clustered between 15 and 40 seconds. This relatively flat clustering is the desired behavior. However, a significant number of workers also spike quite high. No worker stays at the top of the duration chart for the whole experiment, reflecting that our load balancing system effectively detects stragglers and reduces their workload. However, it seems that the system may not be granular enough. When we assign more work to early contributors, they often shoot up well above the desired range, becoming outliers for one round before being reduced back down. Adding a smaller amount of work to early completers could help modulate this pattern in the future.

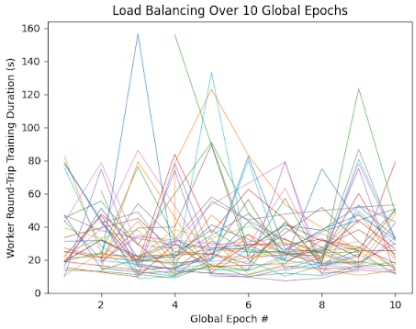

Fig. 4. Round Trip Times over Time

C. Fault Tolerance with N Worker Failures

Due to the distributed nature of the system's target environment, we need to be able to tolerate some level of worker failure. Because there is only one coordinator server, the odds of a critical failure at the coordinator are tolerably low. However, with a large, wide area network, the odds that one or more worker machines will experience an issue at some point during a training run are high. Therefore, we include several fault tolerance mechanisms. If the worker experiences a transient network error or worker-side glitch, it has a built-in retry mechanism that will allow it to reestablish contact with the coordinator server and join back into the next training epoch. However, for this experiment, we completely terminated the worker process on each of N machines, simulating unrecoverable worker failures. For irrecoverable worker failures, the primary fault tolerance mechanism for our system is the quorum protocol. If enough workers remain in the network to complete the current epoch, the epoch can be completed. Afterwards, the failed worker will be removed from the worker list if it does not reconnect during the following epoch. For this experiment, we set the quorum percentage

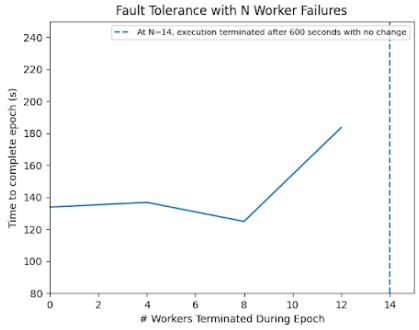

Fig. 5. Time to Complete Epoch

parameter to 75%, then started the coordinator. In each round of experimentation, we waited until the beginning of the second global epoch, when worker membership had stabilized, and then terminated N randomly selected worker processes. Figure 5 shows the time required to complete the subsequent epoch.

The results show that the quorum protocol gives our system good resilience to irrecoverable worker failures until the percentage of failed workers becomes so great that we are no longer able to achieve a quorum. When just a few nodes fail, performance remains steady. Once many nodes begin to fail, the time to complete an epoch begins to increase. When there are many "spare" nodes, an epoch can be completed without waiting for the slowest straggler workers. However, once the number of failed nodes increases, the coordinator needs to wait for most or all nodes to return their results, requiring it to wait for stragglers it previously would not have. Finally, once so many nodes have been terminated that the system is unable to achieve a quorum even once all active nodes have returned and therefore the system fails. This is shown in Figure 5 with a dashed vertical line where with 14 terminated workers, the coordinator was not able to start a new epoch at all.

VI. Conclusions

Our federated learning infrastructure provides reasonable levels of performance, load balancing, and fault tolerance. We demonstrated methods for solving distributed machine learning problems on a wide area network in a way that preserves data privacy by leaving all sensitive training data on user machines. The two way communication between the coordinator server and workers provides a flexible architecture for coordinating federated learning. In our model, workers do not need to send their data to the coordinator and no single node needs any information to start with except for the address of the coordinator. Therefore, this architecture can be applied across multiple domains. For example, it would work equally well for an internal company network spread across multiple locations as it would for a public, crowdsourced model.

Future work on this project could include developing more sensitive load balancing techniques and enhancing the quorum protocol to ensure that the system continues to work well in situations in which training data is very unevenly distributed across workers. Additionally, secure aggregation and differential privacy techniques would improve the security of our system.

References

[1] Bonawitz, Keith, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. B. McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage and Jason Roselander. "Towards Federated Learning at Scale: System Design," March 22, 2019. https://arxiv.org/pdf/1902.01046.pdf

[2] Collins, Liam, Hamed Hassani, Aryan Mokhtari and Sanjay Shakkottai. "FedAvg with Fine Tuning: Local Updates Lead to Representation Learning," May 27, 2022. https://arxiv.org/abs/2205.13692

[3] Li, Li, Yuxi Fan, Mike Tse,and Kuo-Yi Lin."A review of applications in federated learning," Computers & Industrial Engineering. Volume 149, 2020. https://doi.org/10.1016/j.cie.2020.106854.

[4] Li, Qinbin, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yaun Li, Xu Liu, and Bingsheng He. "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection," December 5, 2021. https://arxiv.org/pdf/1907.09693.pdf.

[5] Paulik, Matthias, Matthew Stephen Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier C. van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, Sudeep Agarwal, Julien Freudiger, Andrew Byde, Abhishek Bhowmick, Gaurav Kapoor, Si Beaumont, Áine Cahill, Dominic Hughes, Omid Javidbakht, Fei Dong, Rehan Rishi and Stanley Hung. "Federated Evaluation and Tuning for On-Device Personalization: System Design & Applications," February 16, 2021. https://arxiv.org/pdf/2102.08503.pdf.

[6] Satariano, Adam. "Chatgpt Is Banned in Italy over Privacy Concerns." The New York Times, March 31, 2023. https://www.nytimes.com/2023/03/31/technology/chatgpt-italy-ban.html.

[7] Tajabadi, Mohammad, Linus Grabenhenrich, Adèle Ribeiro, Michael Leyer, and Dominik Heider. "Sharing Data with Shared Benefits: Artificial Intelligence Perspective." Journal of Medical Internet Research, March 23, 2023. https://www.jmir.org/2023/1/e47540.