Federated Learning

Data is born at the edge

Billions of phones & IoT devices constantly generate data

Data enables better products and smarter models

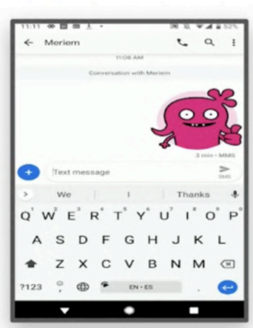
With everyone becoming more privacy aware, we want to be nice if you can train a model without needing to upload highly identifiable personal information to a Cloud hosting server. What if instead, data can be covered anonymous or some of the training can even be done in a distributive way.

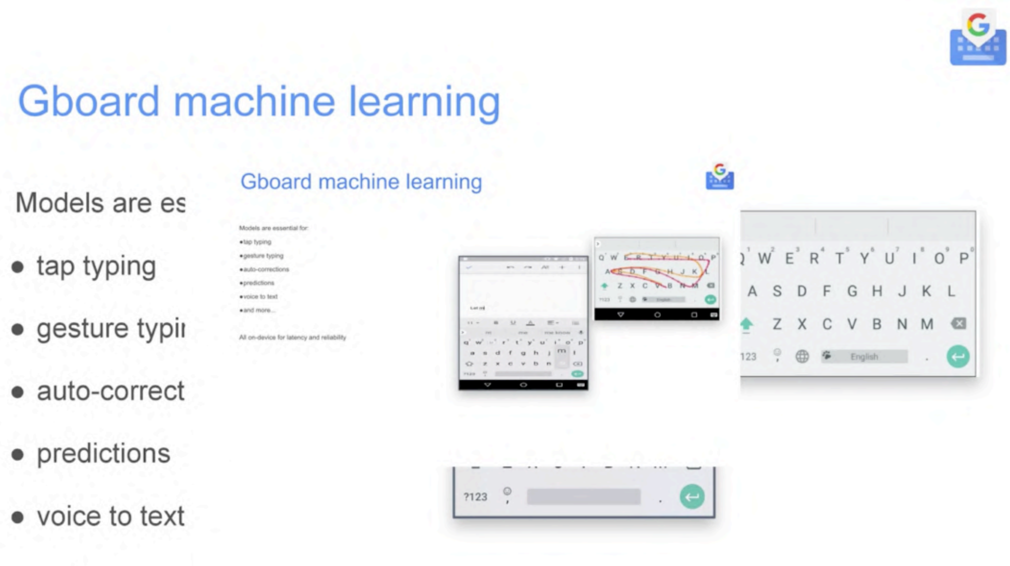
Can data live at the edge?

On-device inference offers:

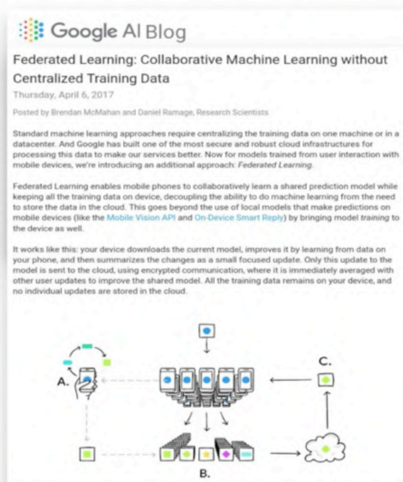
- Improved latency
- Works offline
- Better battery life
- Privacy advantages

Gboard: mobile keyboard





https://ai.googleblog.com/2017/04/federated-learning-collaborative.html



Google AI Blog

Federated Learning: Collaborative Machine Learning without Centralized Training Data

Thursday, April 6, 2017

Posted by Brendan McMahan and Daniel Ramage, Research Scientists

Standard machine learning approaches require centralizing the training data on one machine or in a datacenter. And Google has built one of the most secure and robust cloud infrastructures for processing this data to make our services better. Now for models trained from user interaction with mobile devices, we're introducing an additional approach: *Federated Learning*.

Federated Learning enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on device, decoupling the ability to do machine learning from the need to store the data in the cloud. This goes beyond the use of local models that make predictions on mobile devices (like the Mobile Vision API and On-Device Smart Reply) by bringing model *training* to the device as well.

It works like this: your device downloads the current model, improves it by learning from data on your phone, and then summarizes the changes as a small focused update. Only this update to the model is sent to the cloud, using encrypted communication, where it is immediately averaged with other user updates to improve the shared model. All the training data remains on your device, and no individual updates are stored in the cloud.

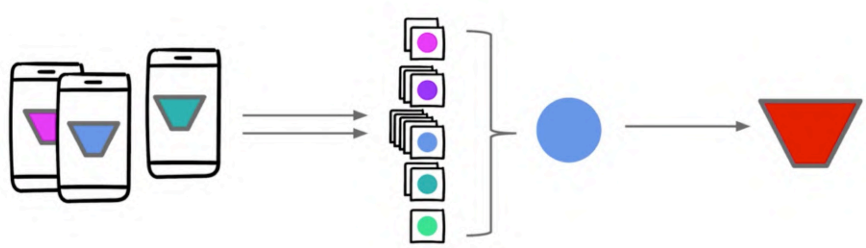
Examples of the learning and not the data used to generate the learning can be shared across all users. So how does all of this work?

Well, let's consider a simple scenario like this one. I have a population of users each with a mobile device and each with my model running on it. If I want to do some training on these devices, I realize that I'm getting them to do something that's computationally expensive.

So I can select a subset of them that are available in the sense that they're not currently being used, they're plugged in, and they're charging and they're idle. The principle here is that I don't want to damage the user experience. Of the available ones, some of them will have data that's relevant to the problem I'm trying to solve like training to understand the keyboard usage better for example. So you'll pick a subset of the available devices.

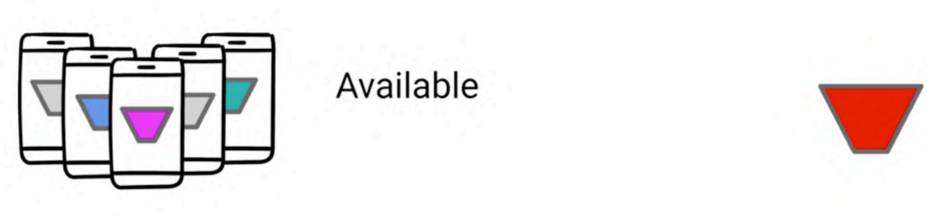
These will then receive a training model which can then be retrained on the device

The results of the training, not the data used to perform the training, is sent to the server.



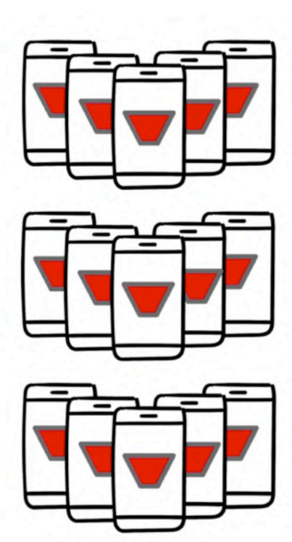
The server then uses this to retrain the master model.

Before deploying the new model to your customers devices, of course, the model should be tested and what better for testing than to use a very similar approach using a population of the user set to test if the model works well with their data.



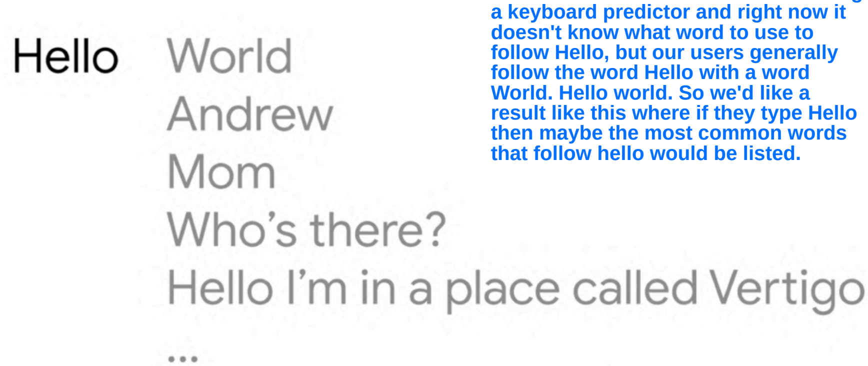
So we can take a subset for testing the same way as we took a subset for training and deploy a new model to those.

Using a process like this and iterating, then over time we can improve the model significantly and the user's personal data never leaves their device. Each device can benefit from the experience of others so the learning is distributed across all users and the term federated is used to describe this.



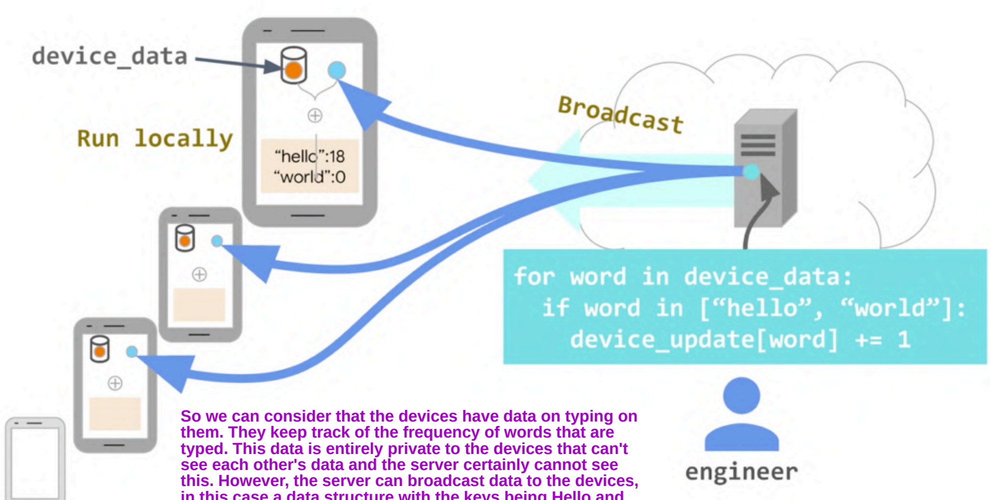
Previously, you had an overview of the concepts of federated learning and how a master model could be retrained with distributed users data without the user's data being uploaded to a server where it could potentially be misused. The data always stays on the user's device. At the end of that demonstration, we realized that there is the potential for the retrained models that the user has uploaded to possibly be reverse engineered to get at that data.

So with that in mind, I'd like to go through two concepts now that can help ease any fears here. The first is that by principle federated learning works only on aggregates and the second to show how the data can be encrypted by the clients on route to the server. First, let's take a look at an aggregation methodology that can be used to maintain an individual user's privacy.



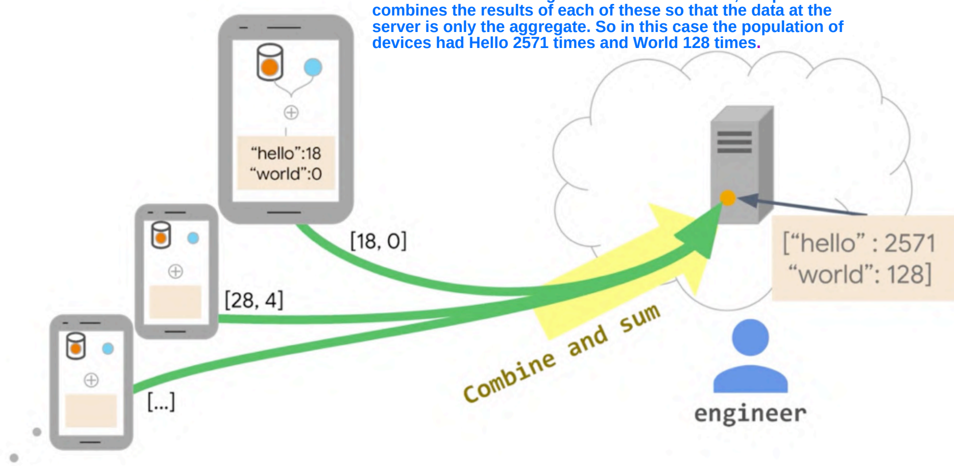
Consider this scenario we're retraining a keyboard predictor and right now it doesn't know what word to use to follow Hello, but our users generally follow the word Hello with a word World. Hello world. So we'd like a result like this where if they type Hello then maybe the most common words that follow hello would be listed.

But let's keep it really simple and just theorize how often World will follow Hello as people type. We think it happens a lot. But let's learn from our users just how often it follows.

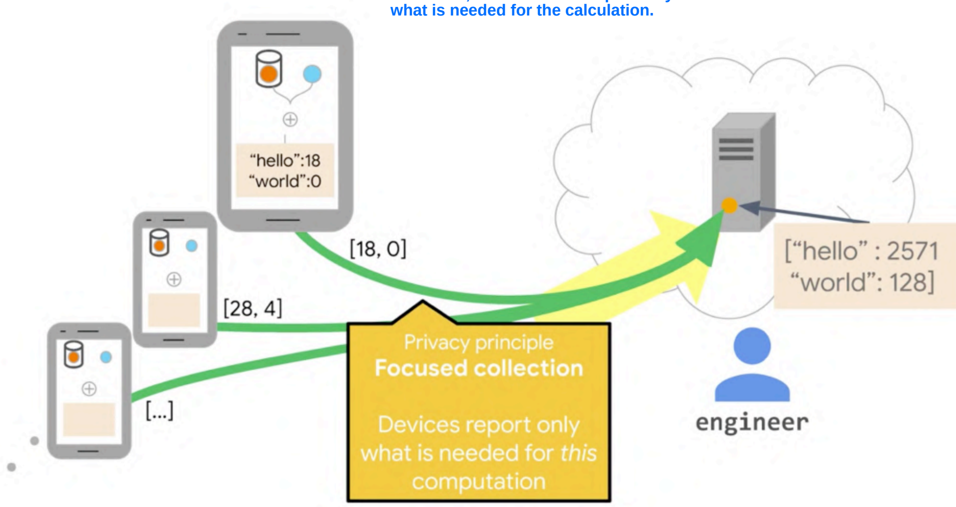


So we can consider that the devices have data on typing on them. They keep track of the frequency of words that are typed. This data is entirely private to the devices that can't see each other's data and the server certainly cannot see this. However, the server can broadcast data to the devices, in this case a data structure with the keys being Hello and World. Each device can then fill in the values for these keys. So for example, the top device here has Hello 18 times but never has World.

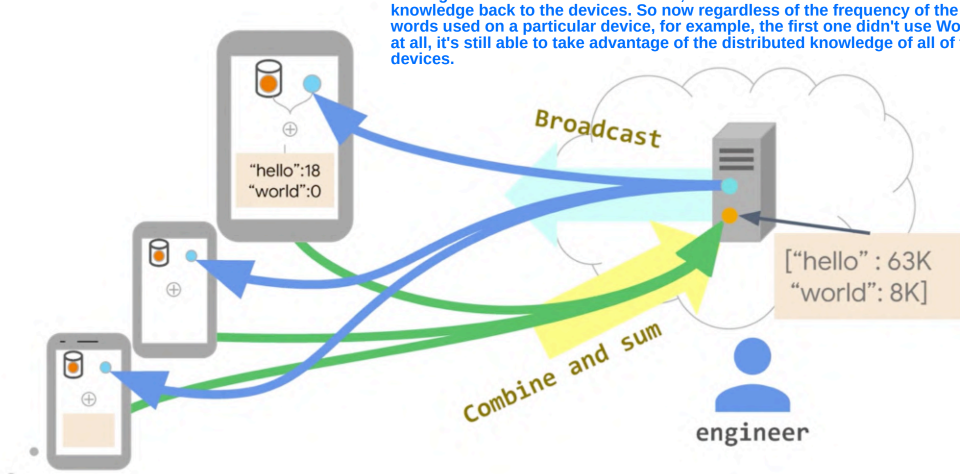
Other devices will have different experiences, of course, where the second may have Hello 28 times and World four times. But instead of each sending its values to the server, the protocol combines the results of each of these so that the data at the server is only the aggregate. So in this case the population of devices had Hello 2571 times and World 128 times.

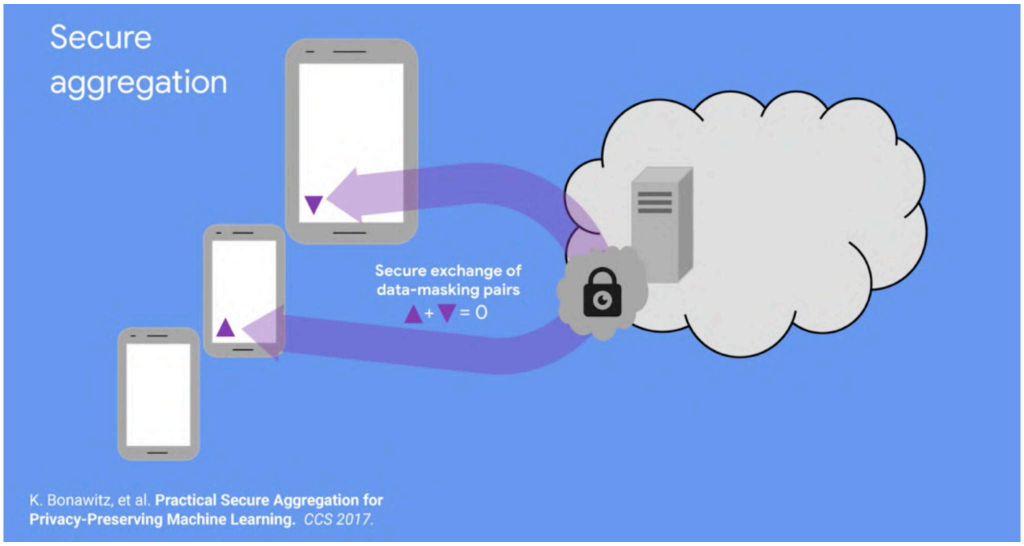


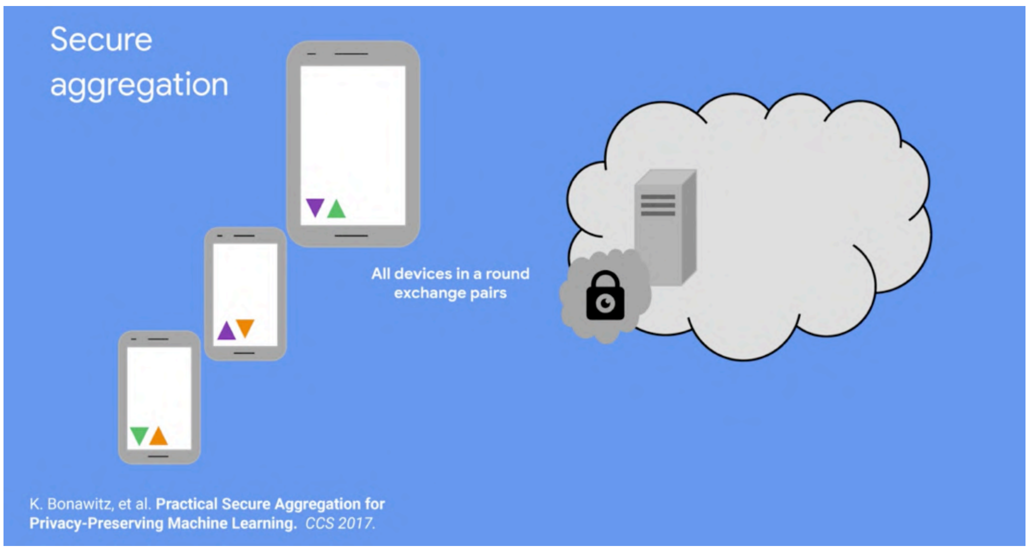
This is a privacy principle called focused collection, where devices report only what is needed for the calculation.

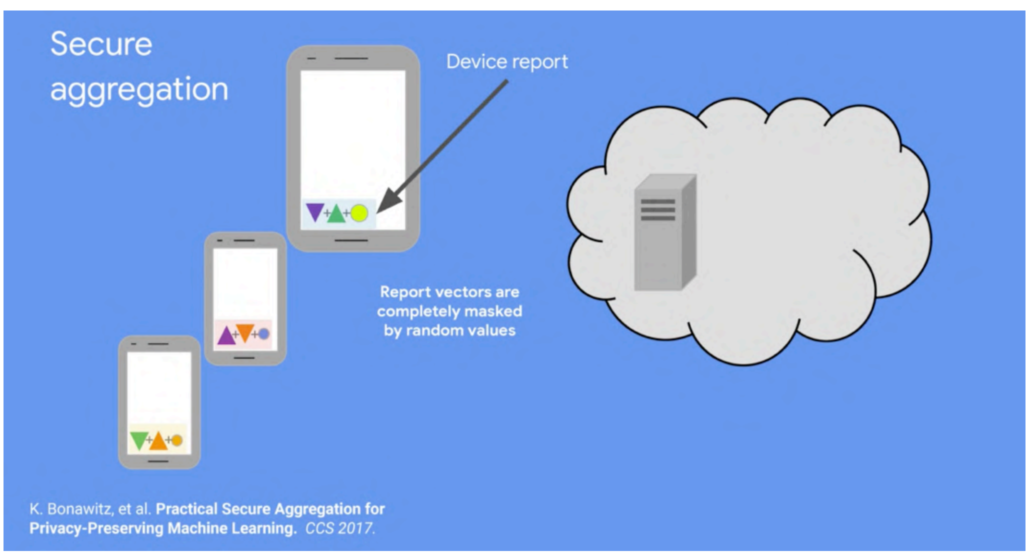


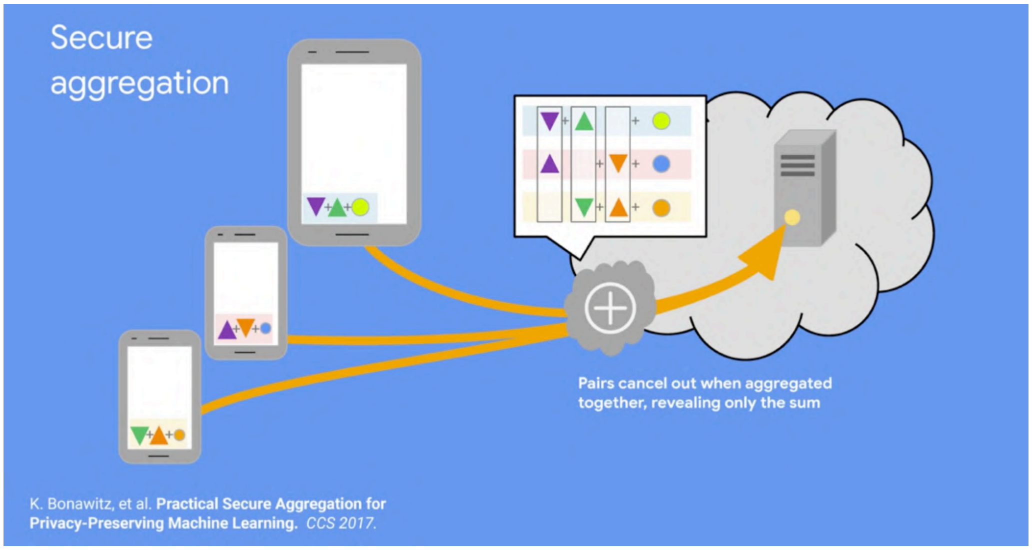
From the data collected then from all devices, the server then finds that for 63,000 instances of Hello, there were 8,000 Instances of World. So roughly one-eighth of the time Hello was used, so was world. It can then distribute this knowledge back to the devices. So now regardless of the frequency of the words used on a particular device, for example, the first one didn't use World at all, it's still able to take advantage of the distributed knowledge of all of the devices.

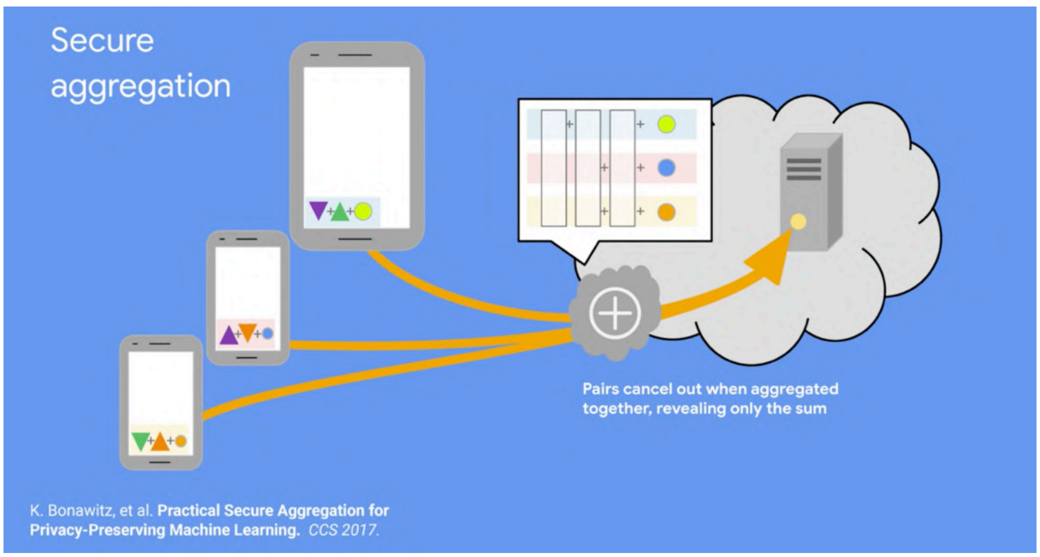



Secure aggregation
Secure exchange of data-masking pairs
+ = 0
K. Bonawitz, et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. CCS 2017.


Secure aggregation
All devices in a round exchange pairs
K. Bonawitz, et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. CCS 2017.


Secure aggregation
Device report
Report vectors are completely masked by random values
K. Bonawitz, et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. CCS 2017.

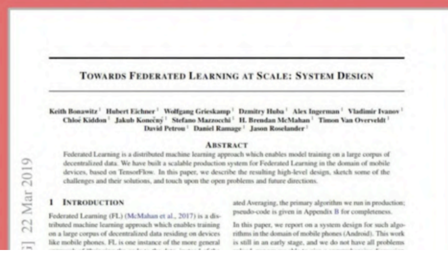

Secure aggregation
Pairs cancel out when aggregated together, revealing only the sum
K. Bonawitz, et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. CCS 2017.


Secure aggregation
Pairs cancel out when aggregated together, revealing only the sum
K. Bonawitz, et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. CCS 2017.


https://eprint.iacr.org/2017/281.pdf
Practical Secure Aggregation
for Privacy-Preserving Machine Learning
Keith Bonawitz*, Vladimir Ivanov*, Ben Kreuter*,
Antonio Marcedone†‡, H. Brendan McMahan*, Sarvar Patel*,
Daniel Ramage*, Aaron Segal*, and Karn Seth*
*{bonawitz,vlivan,benkreuter,mcmahan,
sarvar,dramage,asegal,karn}@google.com
Google, Mountain View, CA 94043
‡marcedone@cs.cornell.edu
Cornell Tech, 2 West Loop Rd., New York, NY 10044
1 INTRODUCTION
1.1 Our Results
1.2 Organization

Google's federated system

Towards Federated Learning at Scale: System Design
ai.google/research/pubs/pub47976



K. Bonawitz, *et al*. **Towards Federated Learning at Scale: System Design**. *SysML 2019*.

tensorflow.org/federated

TensorFlow Federated

What's in the box

Federated Learning (FL) API

- Implementations of federated training/evaluation
- Can be applied to existing TF models/data

Federated Core (FC) API

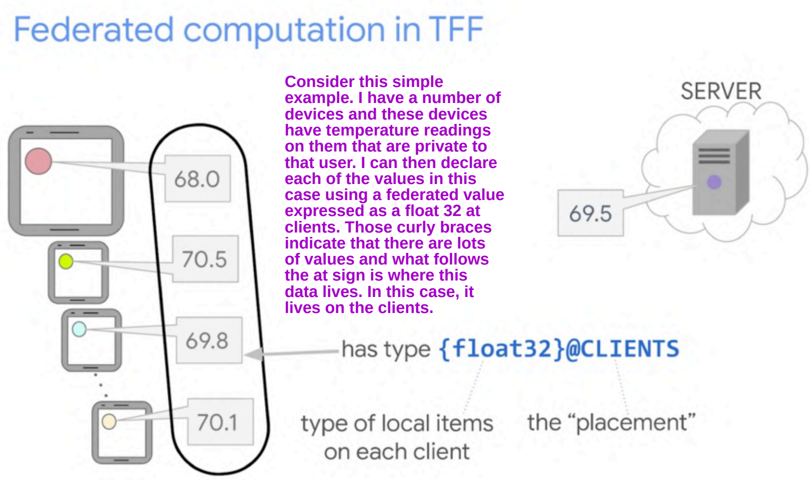
- Allows for expressing new federated algorithms

Local runtime for simulations

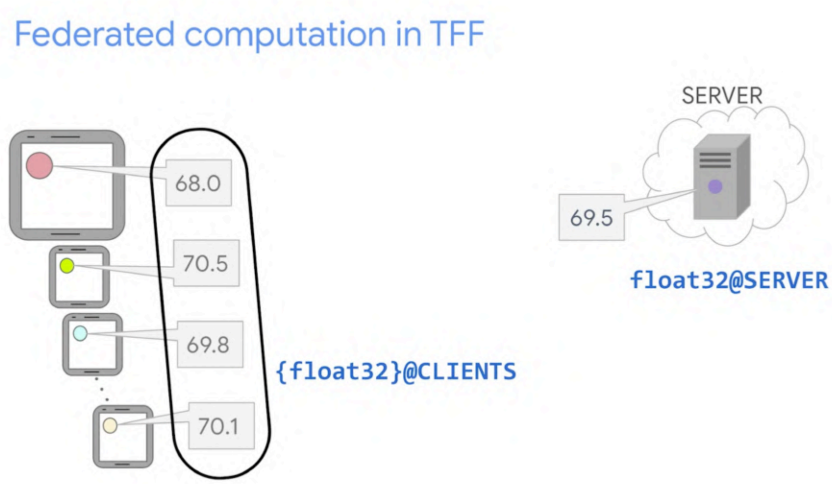
TensorFlow Federated offers two main APIs for federated learning. The federated learning API contains implementations of federated training and evaluation that can be applied to existing care as models. So you can experiment with learning using them.

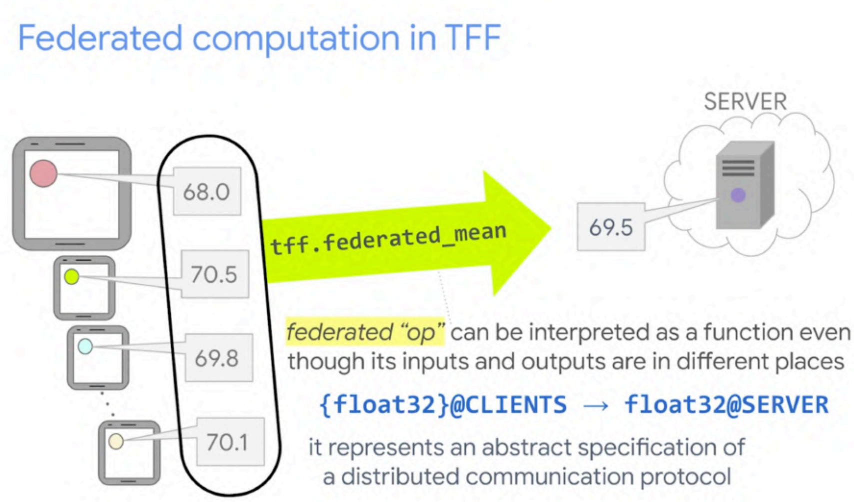
Note that there are no more mobile APIs yet. Everything is done in simulation and the collabs at the TensorFlow site will demonstrate this.

Similarly, the federated core API allows you to express new federated algorithms and to test them in a simulated environment.



Consider this simple example. I have a number of devices and these devices have temperature readings on them that are private to that user. I can then declare each of the values in this case using a federated value expressed as a float 32 at clients. Those curly braces indicate that there are lots of values and what follows the at sign is where this data lives. In this case, it lives on the clients.





We've seen already that our principle is to aggregate data prior to sending it to the server and the API implements this via tff.federated_mean. This operation can be interpreted as a single function, even though its inputs and outputs are in vastly different places. We call this a federated operation. Federated ops like this represent an abstract implementation of the distributed communication protocol. You don't write the code for handling all of the communication yourself. There are a number of these federated operations that represent the common building blocks for any federated learning system.

Federated computations in TFF

```
READINGS_TYPE = tff.FederatedType(tf.float32, tff.CLIENTS)

# An abstract specification of a simple distributed system
@tff.federated_computation(READINGS_TYPE)
def get_average_temperature(sensor_readings):
  return tff.federated_mean(sensor_readings)
```

Let's look at a brief code example using TFF. I'm not going to go to into depth so it might look a little confusing, but the TensorFlow site has a colab that you can walk through that implements all of this. We'll first implement a federated type that represents inputs. Note that we specify that it's on our clients.

Federated computations in TFF

```
READINGS_TYPE = tff.FederatedType(tf.float32, tff.CLIENTS)

# An abstract specification of a simple distributed system
@tff.federated_computation(READINGS_TYPE)
def get_average_temperature(sensor_readings):
  return tff.federated_mean(sensor_readings)
```

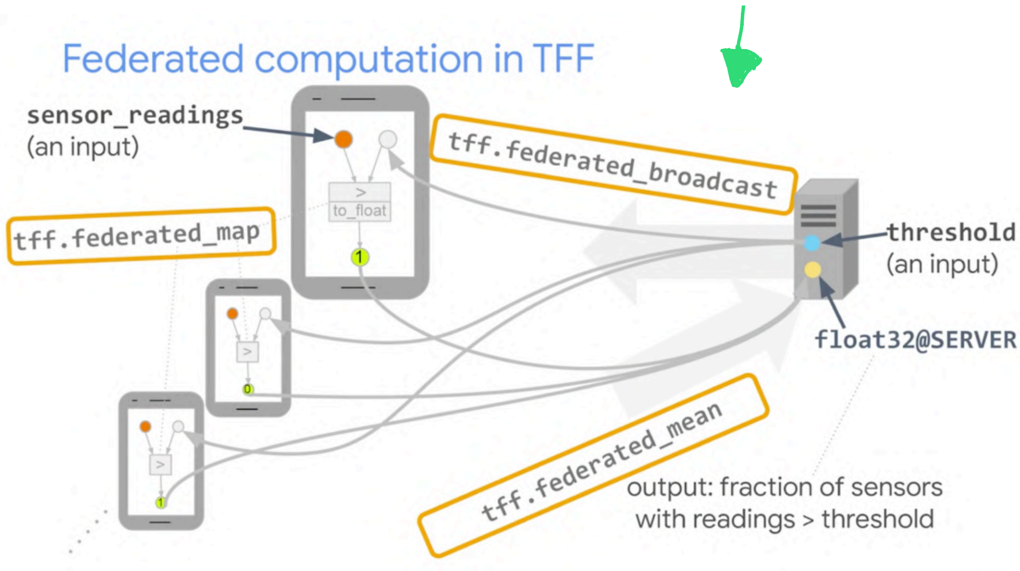
We're then going to implement a function to get the average of the sensor readings. But in order for this to be federated, we'll use a function decorator saying that it's a federated computation using @tff.federated_computation

Federated computations in TFF

```
READINGS_TYPE = tff.FederatedType(tf.float32, tff.CLIENTS)

# An abstract specification of a simple distributed system
@tff.federated_computation(READINGS_TYPE)
def get_average_temperature(sensor_readings):
  return tff.federated_mean(sensor_readings)
```

In this function I'll return tff.federated_mean of the sensor readings. Under the hood, the readings are being gathered by the server using the secure protocols, decrypted, and then returns to me as an aggregates.

Similarly, if we want to do a federated computation, we will broadcasts from the server a value that we want to learn about. For example, if the sensor readings might be above a certain value. We can do this with tff.federated_broadcasts. In this case, we want to find the number of machines where the value of their temperature is above a certain amount, say 70 degrees. The server will broadcast the data and the device will check it against its internal value giving us a one if it's higher and a zero if it isn't. These values are then aggregated using a federated mean and the server can find out how many of the devices are above that value overall without it ever knowing

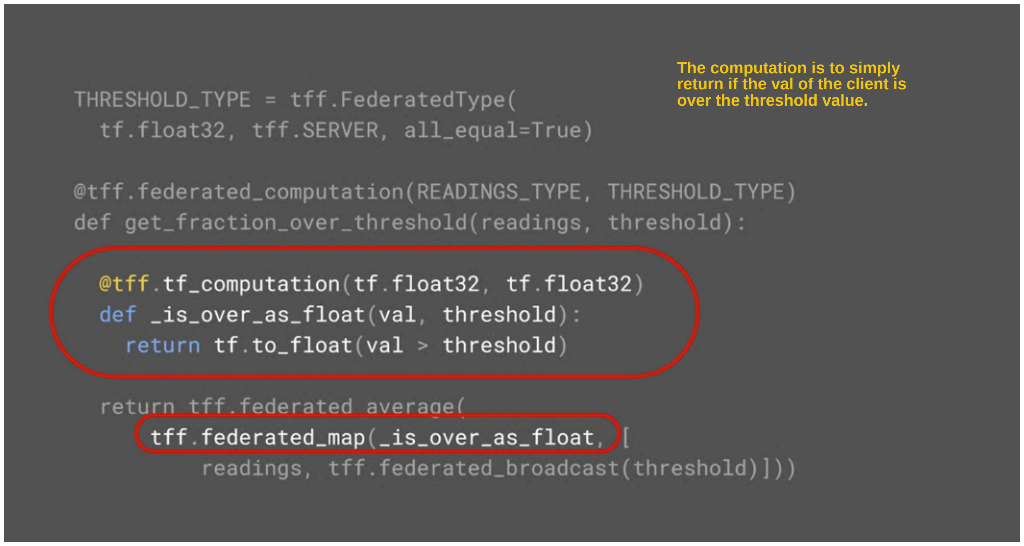


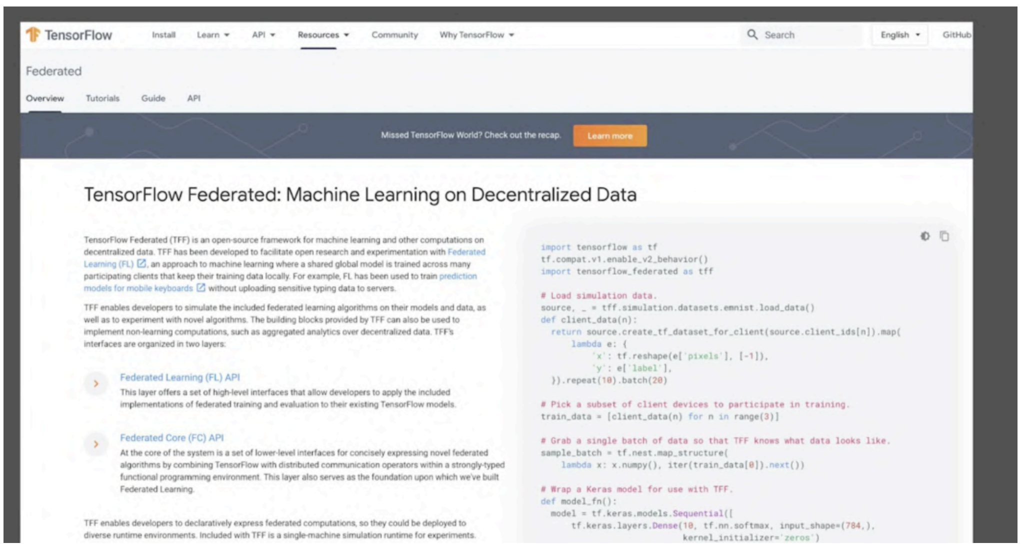

```python
THRESHOLD_TYPE = tff.FederatedType(
  tf.float32, tff.SERVER, all_equal=True)

@tff.federated_computation(READINGS_TYPE, THRESHOLD_TYPE)
def get_fraction_over_threshold(readings, threshold):

  @tff.tf_computation(tf.float32, tf.float32)
  def _is_over_as_float(val, threshold):
    return tf.to_float(val > threshold)

  return tff.federated_average(
      tff.federated_map(_is_over_as_float, [
          readings, tff.federated_broadcast(threshold)]))
```

The code for this is very similar. We'll create a federated type for a threshold and we'll indicate that it lives on the server with tff.server. We'll then decorate a function called get_fraction_over_threshold as a federated computation.

```python
THRESHOLD_TYPE = tff.FederatedType(
  tf.float32, tff.SERVER, all_equal=True)

@tff.federated_computation(READINGS_TYPE, THRESHOLD_TYPE)
def get_fraction_over_threshold(readings, threshold):

  @tff.tf_computation(tf.float32, tf.float32)
  def _is_over_as_float(val, threshold):
    return tf.to_float(val > threshold)

  return tff.federated_mean(
      tff.federated_map(_is_over_as_float, [
          readings, tff.federated_broadcast(threshold)]))
```

This function will return a federated mean which takes the readings and the thresholds that was broadcast as inputs to a federated map value





https://www.youtube.com/watch?v=89BGjQYA0uE