EE-508: Hardware Foundations for Machine Learning

Federated Learning

University of Southern California

Ming Hsieh Department of Electrical and Computer Engineering

Instructors:
Arash Saifhashemi

Federated Learning Tutorial

NeurIPS 2020

*Some enhancements added to the slides for EE538

Peter Kairouz

Brendan McMahan

Virginia Smith

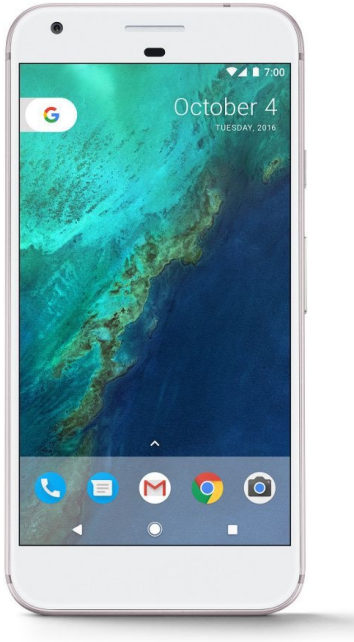
Presenting the work of many

Part I: What is Federated Learning?

Data is born at the edge

Billions of phones & IoT devices constantly generate data

Data enables better products and smarter models

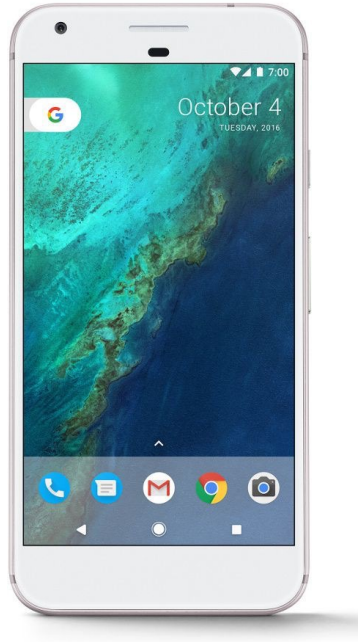


Can data live at the edge?

Data processing is moving on device:

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

E.g., on-device inference for mobile keyboards and cameras.



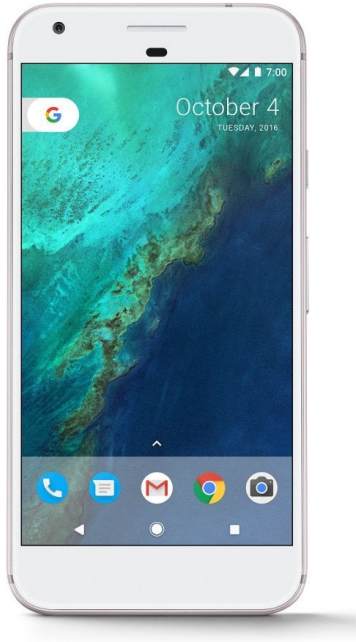
Can data live at the edge?

Data processing is moving on device:

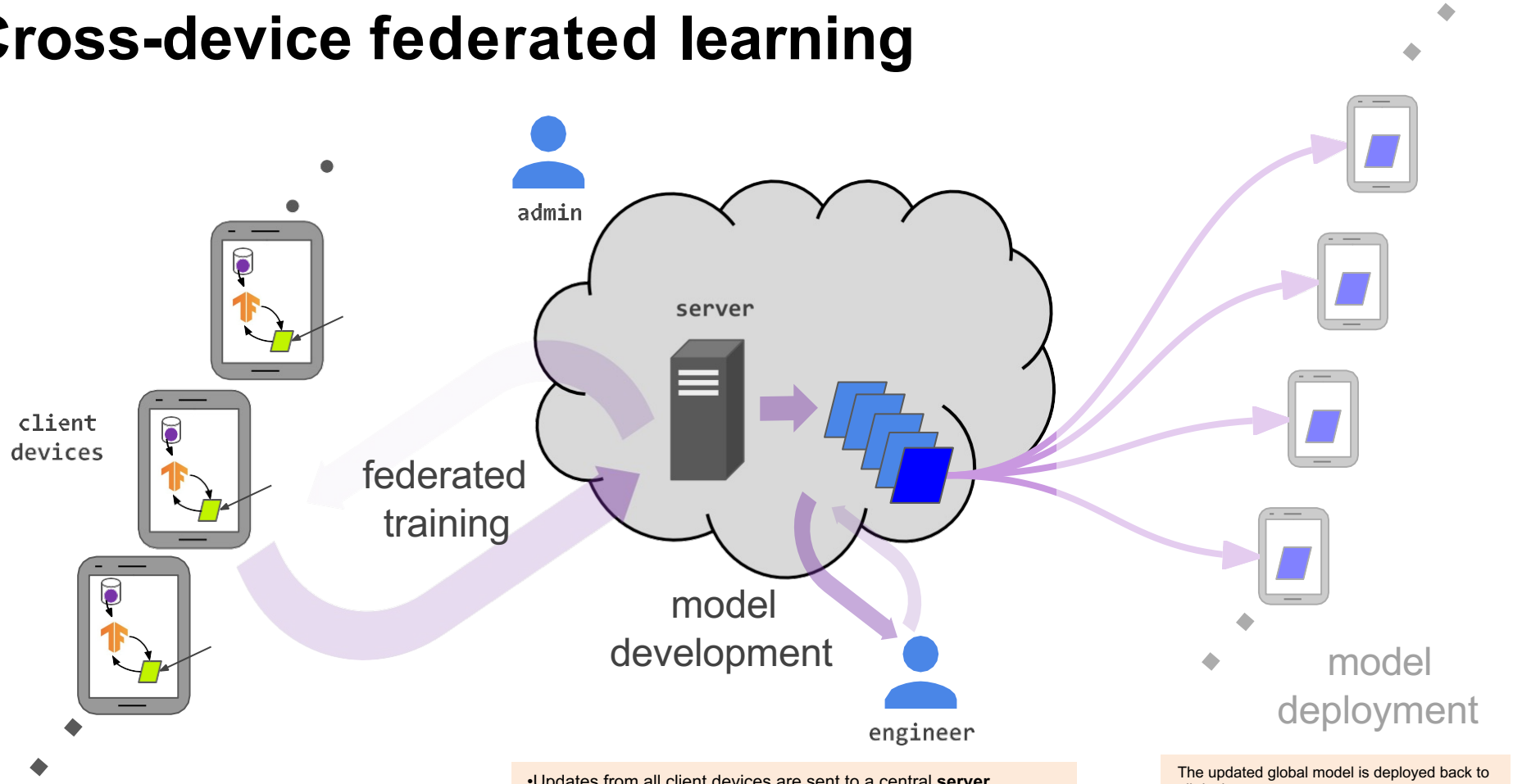
- Improved latency
- Works offline
- Better battery life
- Privacy advantages

E.g., on-device inference for mobile keyboards and cameras.

What about analytics?
What about learning?



Cross-device federated learning



Multiple smartphones with local data.
Each device trains a local model on its private data (e.g. user activity).
No raw data is shared — only model updates are sent.

- Updates from all client devices are sent to a central **server**.
- The **server aggregates** these updates to improve a **global model**.
- The global model improves over time through this process.

The updated global model is deployed back to all devices.
This new model benefits all users without compromising their privacy.

Applications of cross-device federating learning

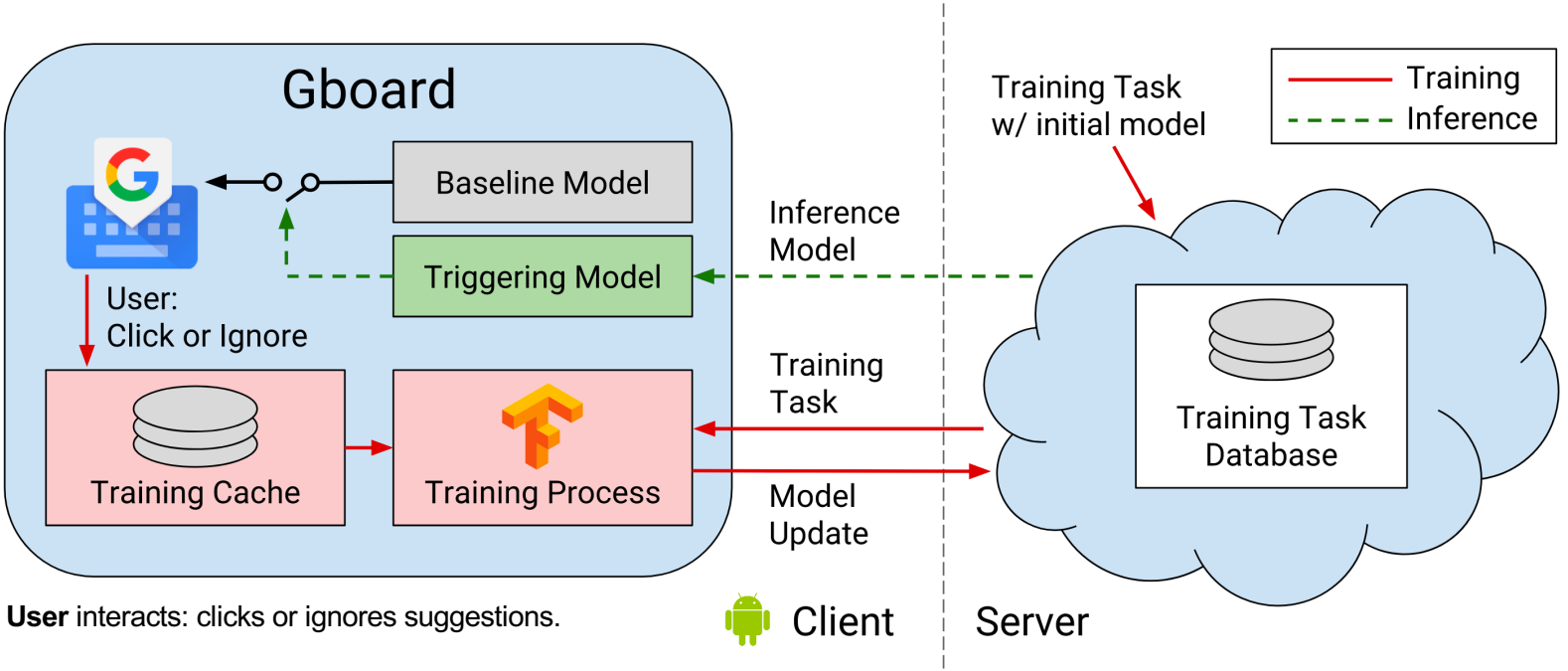
What makes a good application?

- On-device data is more relevant than server-side proxy data
- On-device data is privacy sensitive or large
- Labels can be inferred naturally from user interaction

Example applications

- Language modeling for mobile keyboards and voice recognition
- Image classification for predicting which photos people will share
- ...

Google Android Keyboard



Architecture overview. Inference and training are on-device; model updates are sent to the server during training rounds and trained models are deployed manually to clients.

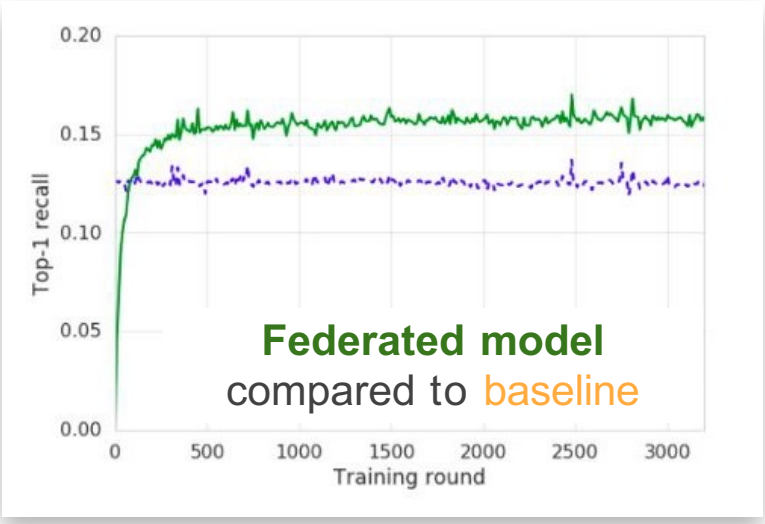
- Improve suggestions without sending sensitive typing data to Google.
- Train models **privately** and **securely** across millions of devices.

Source: Google

Gboard: next-word prediction

Federated RNN (compared to prior n-gram model):

- Better next-word prediction accuracy: +24%
- More useful prediction strip: +10% more clicks



A. Hard, *et al*. **Federated Learning for Mobile Keyboard Prediction**. *arXiv:1811.03604*

Other federated models in Gboard

Emoji prediction

- 7% more accurate emoji predictions
- prediction strip clicks +4% more
- 11% more users share emojis!

Ramaswamy, *et al.* **Federated Learning for Emoji Prediction in a Mobile Keyboard.** arXiv:1906.04329.

Action prediction

When is it useful to suggest a gif, sticker, or search query?

- 47% reduction in unhelpful suggestions
- increasing overall emoji, gif, and sticker shares

T. Yang, *et al*. **Applied Federated Learning: Improving Google Keyboard Query Suggestions.** *arXiv:1812.02903*

Discovering new words

Federated discovery of what words people are typing that Gboard doesn't know.

M. Chen, *et al*. **Federated Learning Of Out-Of-Vocabulary Words.** *arXiv:1903.10635*

Cross-device federated learning at Apple

MIT Technology Review

Sign in Subscribe

Artificial intelligence / Machine learning

How Apple personalizes Siri without hoovering up your data

The tech giant is using privacy-preserving machine learning to improve its voice assistant while keeping your data on your phone.

by **Karen Hao** December 11, 2019

*"Instead, **it relies primarily on a technique called federated learning**, Apple's head of privacy, Julien Freudiger, told an audience at the Neural Processing Information Systems conference on December 8. Federated learning is a privacy-preserving machine-learning method that was first introduced by Google in 2017. It allows Apple to train different copies of a speaker recognition model across all its users' devices, using only the audio data available locally. It then sends just the updated models back to a central server to be combined into a master model. In this way, raw audio of users' Siri requests never leaves their iPhones and iPads, but the assistant continuously gets better at identifying the right speaker."*

https://www.technologyreview.com/2019/12/11/131629/apple-ai-personalizes-siri-federated-learning/

Federated Learning

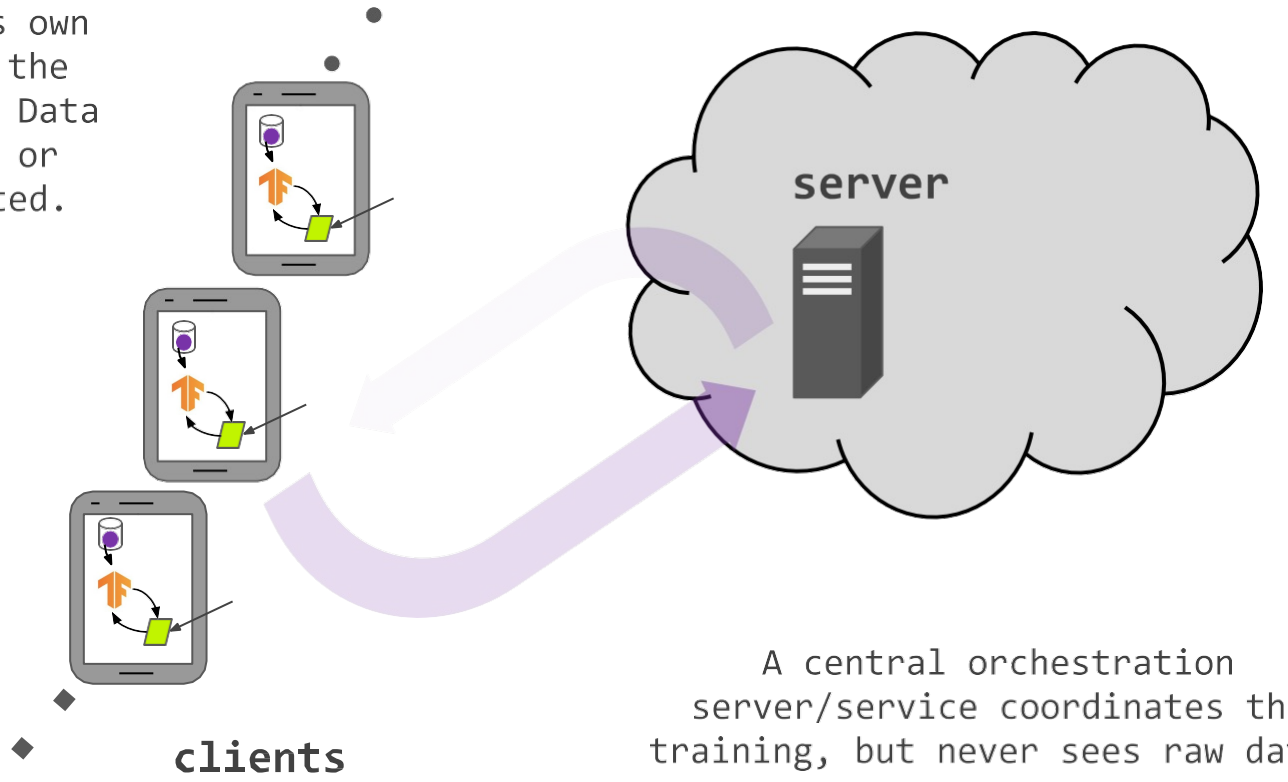
Federated learning is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider.

Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective.

definition proposed in
Advances and Open Problems in Federated Learning (arxiv/1912.04977)

Federated learning - defining characteristics

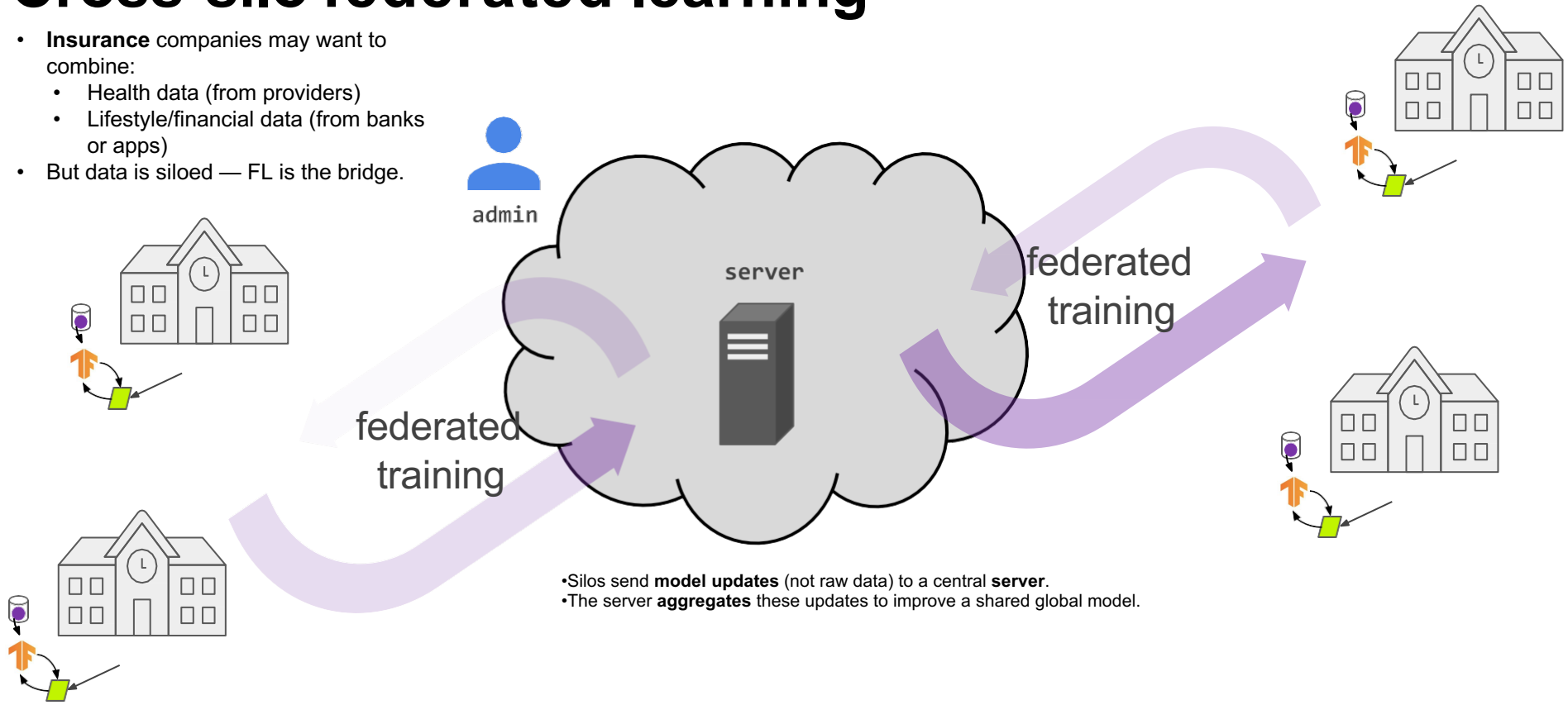
Data is generated locally and remains decentralized. Each client stores its own data and cannot read the data of other clients. Data is not independently or identically distributed.



A central orchestration server/service coordinates the training, but never sees raw data.

Cross-silo federated learning

- **Insurance** companies may want to combine:
  - Health data (from providers)
  - Lifestyle/financial data (from banks or apps)
- But data is siloed — FL is the bridge.



•Silos send **model updates** (not raw data) to a central **server**.
•The server **aggregates** these updates to improve a shared global model.

Data Silos: organizations like hospitals, schools, or banks

Cross-silo = a **small number of reliable, large-scale entities** (e.g., institutions).

Cross-silo federated learning from Intel

ARTIFICIAL INTELLIGENCE, DIAGNOSTICS

UPenn, Intel partner to use federated learning AI for early brain tumor detection

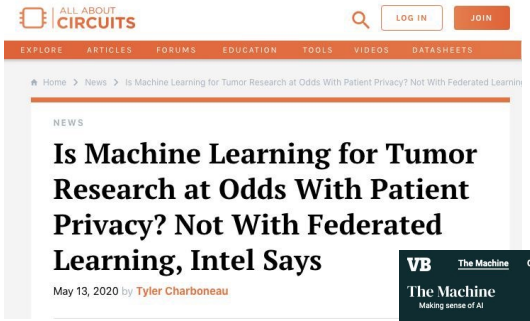
The project will bring in 29 institutions from North America, Europe and India and will use privacy-preserved data to train AI models. Federated learning has been described as being born at the intersection of AI, blockchain, edge computing and the Internet of Things.

By ALARIC DEARMENT

Post a comment / May 11, 2020 at 10:03 AM

"The University of Pennsylvania and chipmaker Intel are forming a partnership to enable 29 heatlhcare and medical research institutions around the world to train artificial intelligence models to detect brain tumors early."

"The program will rely on a technique known as federated learning, which enables institutions to collaborate on deep learning projects without sharing patient data. The partnership will bring in institutions in the U.S., Canada, U.K., Germany, Switzerland and India. The centers – which include Washington University of St. Louis; Queen's University in Kingston, Ontario; University of Munich; Tata Memorial Hospital in Mumbai and others – will use Intel's federated learning hardware and software."



ALL ABOUT CIRCUITS

LOG IN JOIN

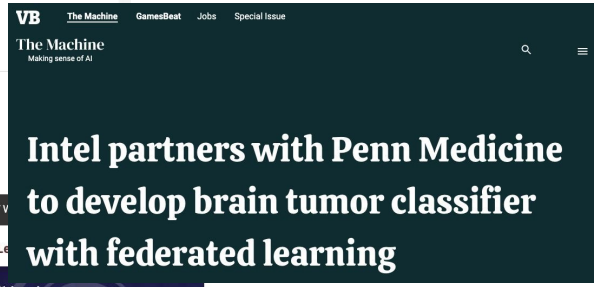
EXPLORE ARTICLES FORUMS EDUCATION TOOLS VIDEOS DATASHEETS

Home > News > Is Machine Learning for Tumor Research at Odds With Patient Privacy? Not With Federated Learning

NEWS

Is Machine Learning for Tumor Research at Odds With Patient Privacy? Not With Federated Learning, Intel Says

May 13, 2020 by Tyler Charboneau



VB The Machine GamesBeat Jobs Special Issue

The Machine
Making sense of AI

Intel partners with Penn Medicine to develop brain tumor classifier with federated learning

Bio·IT World
Next-Gen Technology • Big Data • Personalized Medicine

Subscribe News Advertise Free Downloads Events About Bio-IT W

RELATED STORIES

No Cytokine Storm, 'Bursty' Disease Spread: Biomarkers For Elevated Risk: COVID-19 Updates | Sep 10, 2020

Beyond the Rule of 5: Vastly Expanding Targetable Drugs | Sep 08, 2020

Computational Tool Could Improve Clinical Success Rate Of Drugs | Sep 08, 2020

Mouse Models, Machine Learning, Triggering Proteins: COVID-19 Updates | Sep 03, 2020

Truth Challenge v2: Latest Challenge Results From Genome In A Bottle | Sep

Intel, Penn Medicine Launch Federated Le

By Allison Proffitt

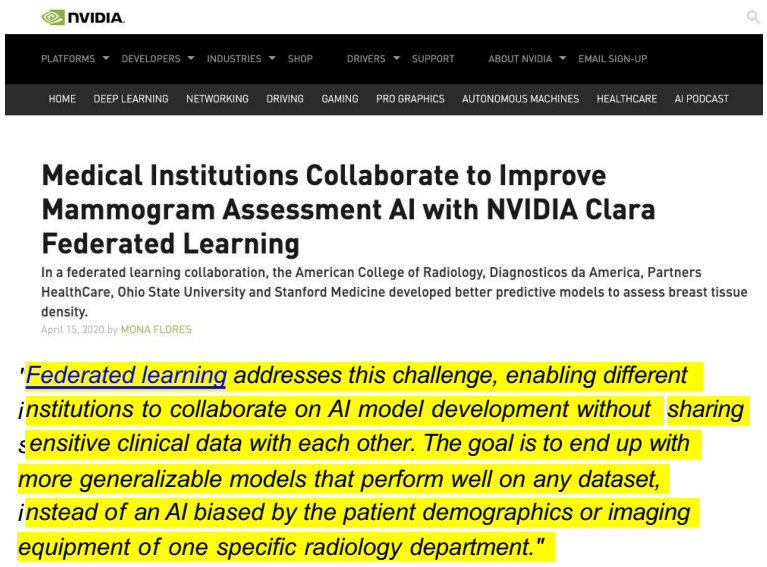
May 28, 2020 | The University of Pennsylvania and Intel have built a federation of 30 institutions to use federated learning to train artificial intelligence (AI) models to identify boundaries of brain tumors.

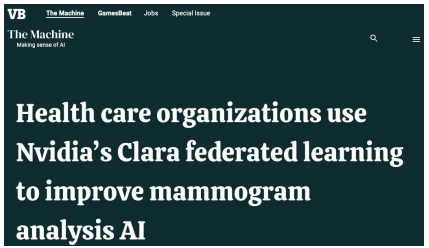
Led by Spyridon Bakas at the Center for Biomedical Image Computing and Analytics (CBICA) at the Perelman School of Medicine at the University of Pennsylvania, the federation is the next step forward in a years-long effort to gather data that would empower AI in brain image analysis.

18th Annual
Discovery on TARGET
VIRTUAL
The Industry's Preeminent Event on Novel Drug Targets and Technologies
September 16-18, 2020

1 https://medcitynews.com/2020/05/upenn-intel-partner-to-use-federated-learning-ai-for-early-brain-tumor-detection/
2 https://www.allaboutcircuits.com/news/can-machine-learning-keep-patient-privacy-for-tumor-research-intel-says-yes-with-federated-learning/
3 https://venturebeat.com/2020/05/11/intel-partners-with-penn-medicine-to-develop-brain-tumor-classifier-with-federated-learning/
4 http://www.bio-itworld.com/2020/05/28/intel-penn-medicine-launch-federated-learning-model-for-brain-tumors.aspx

Cross-silo federated learning from NVIDIA

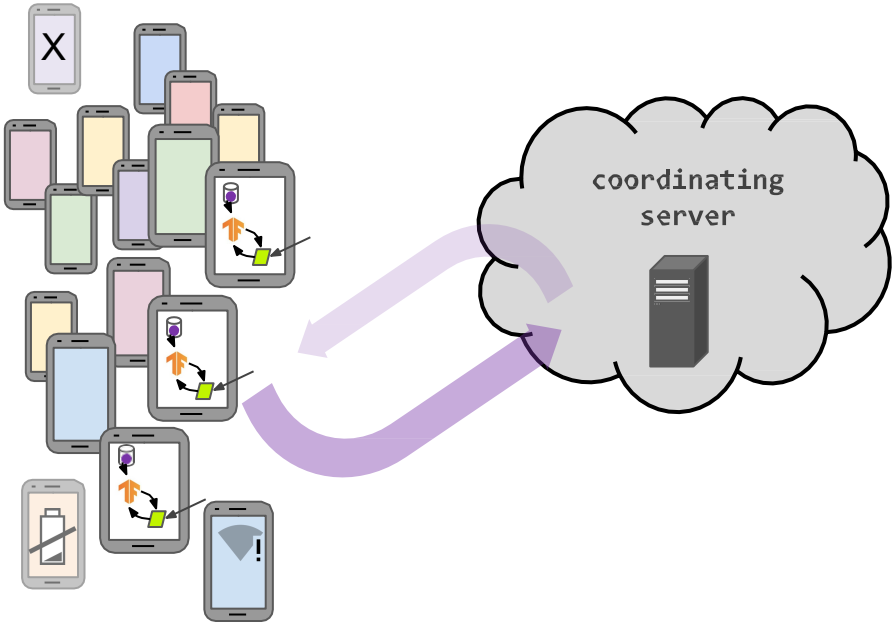




1 https://blogs.nvidia.com/blog/2020/04/15/federated-learning-mammogram-assessment/
2 https://venturebeat.com/2020/04/15/healthcare-organizations-use-nvidias-clara-federated-learning-to-improve-mammogram-analysis-ai/
3 https://medcitynews.com/2020/01/nvidia-says-it-has-a-solution-for-healthcares-data-problems/
4 https://venturebeat.com/2020/06/23/nvidia-and-mercedes-benz-detail-self-driving-system-with-automated-routing-and-parking/

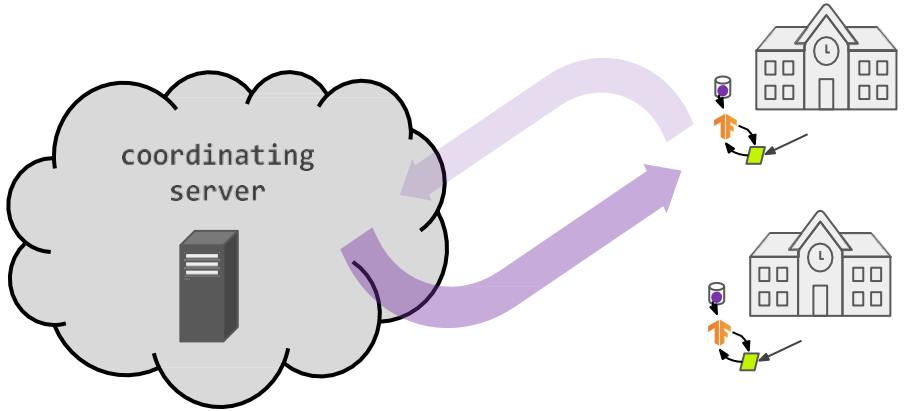
Cross-device federated learning

millions of intermittently available client devices



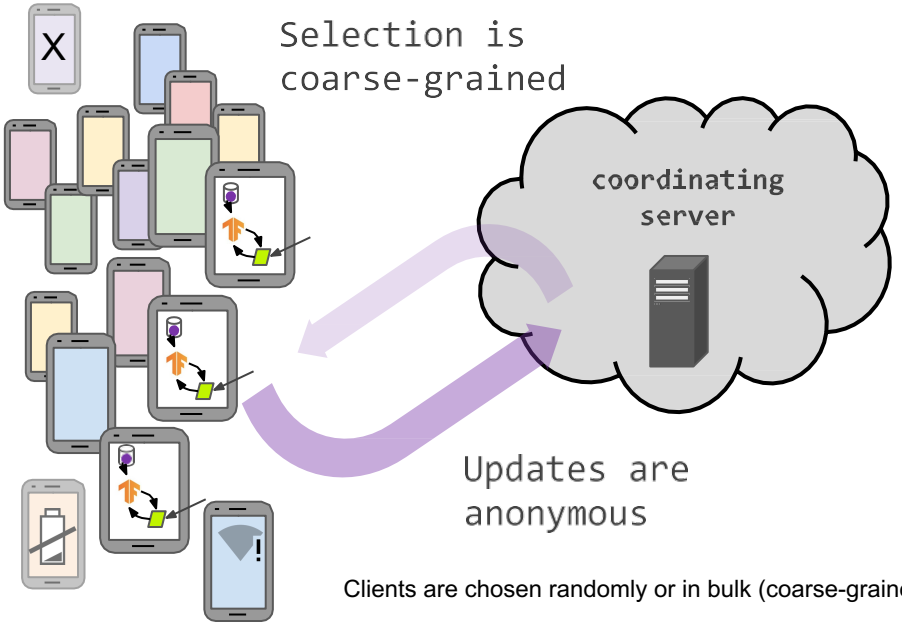
Cross-silo federated learning

small number of clients (institutions, data silos), high availability



Cross-device federated learning

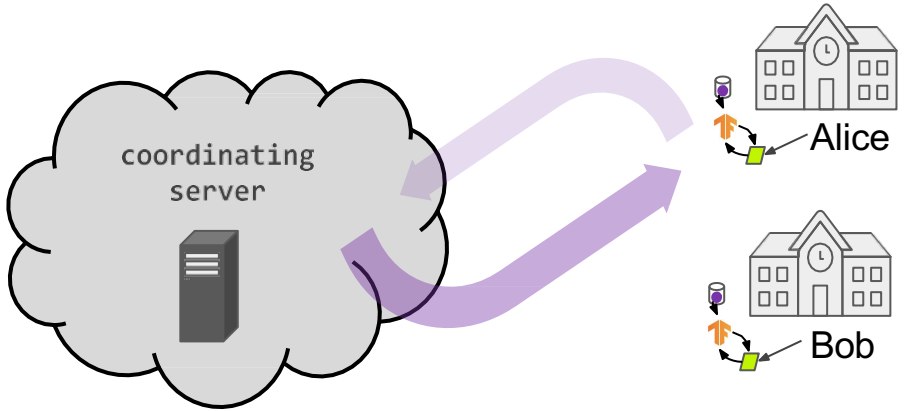
clients cannot be indexed directly (i.e., no use of client identifiers)



Clients are chosen randomly or in bulk (coarse-grained).

Cross-silo federated learning

each client has an identity or name that allows the system to access it specifically

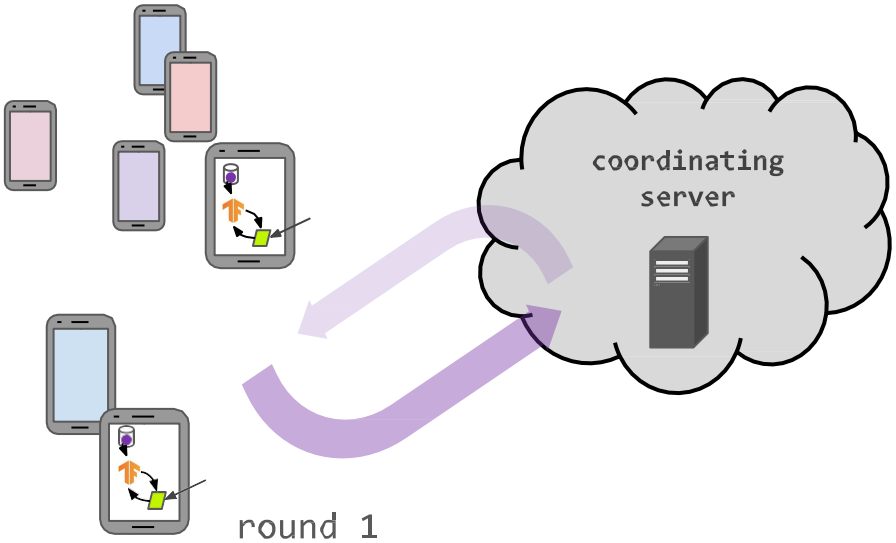


Selection: Targeted — specific clients can be directly contacted.
•**Privacy**: Still preserved, but participants are trusted and known.
•**Stability**: More stable systems and participation.

Cross-device federated learning

Server can only access a (possibly biased) random sample of clients on each round.

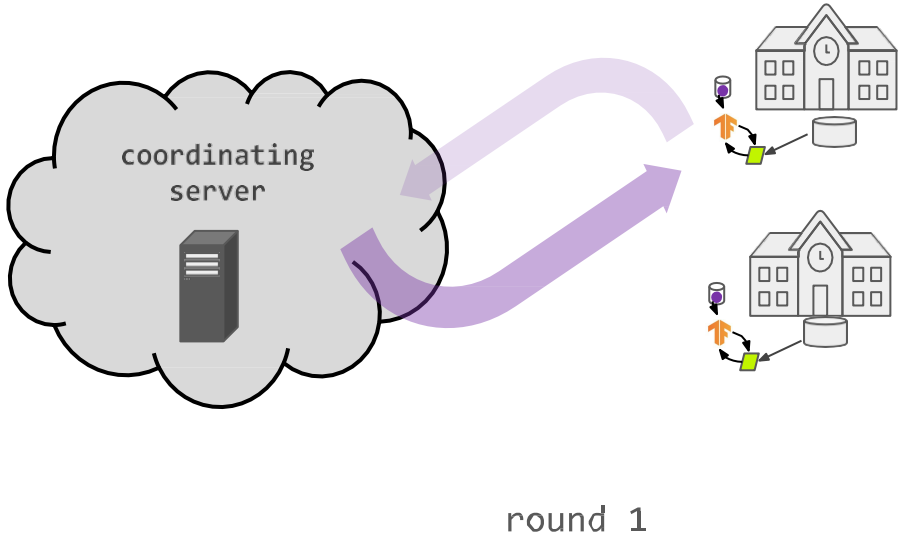
Large population => most clients only participate once.



Cross-silo federated learning

Most clients participate in every round.

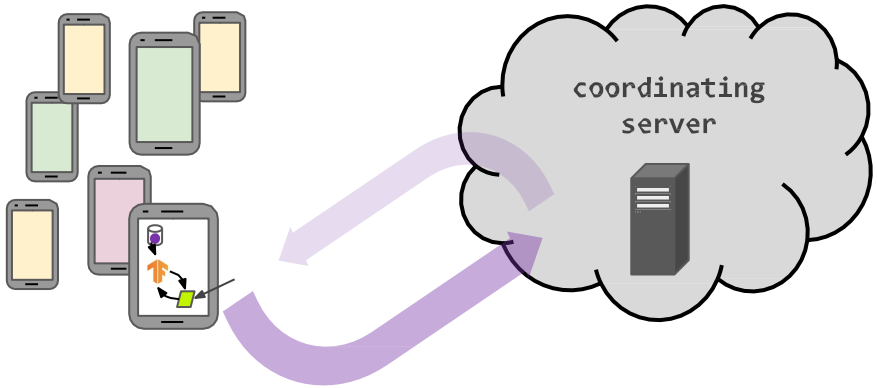
Clients can run algorithms that maintain local state across rounds.



Cross-device federated learning

Server can only access a (possibly biased) random sample of clients on each round.

Large population => most clients only participate once.

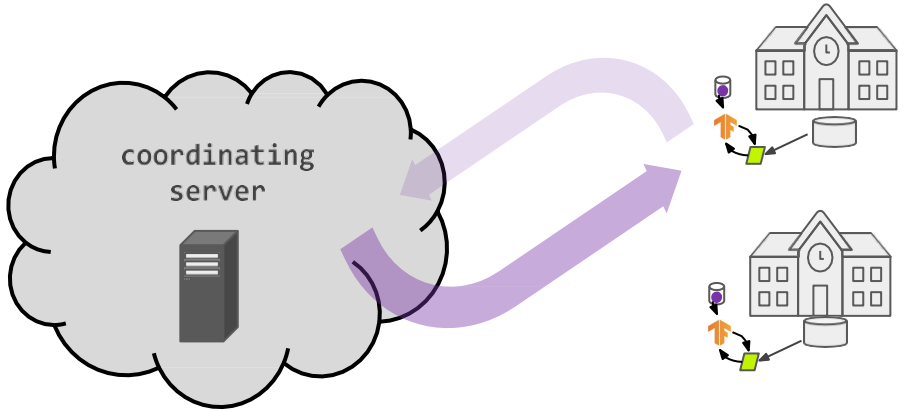


round 2
(completely new set of devices participate)

Cross-silo federated learning

Most clients participate in every round.

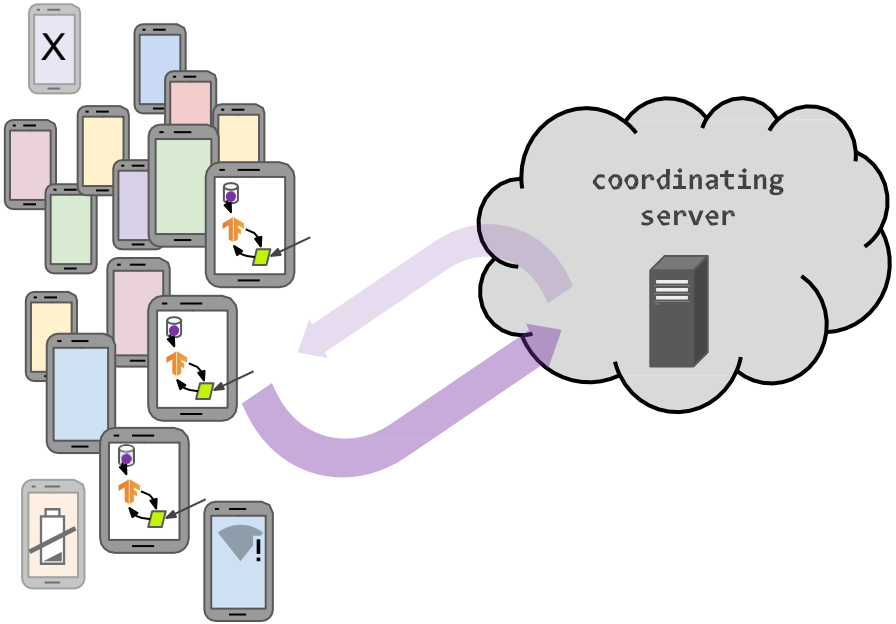
Clients can run algorithms that maintain local state across rounds.



round 2
(same clients)

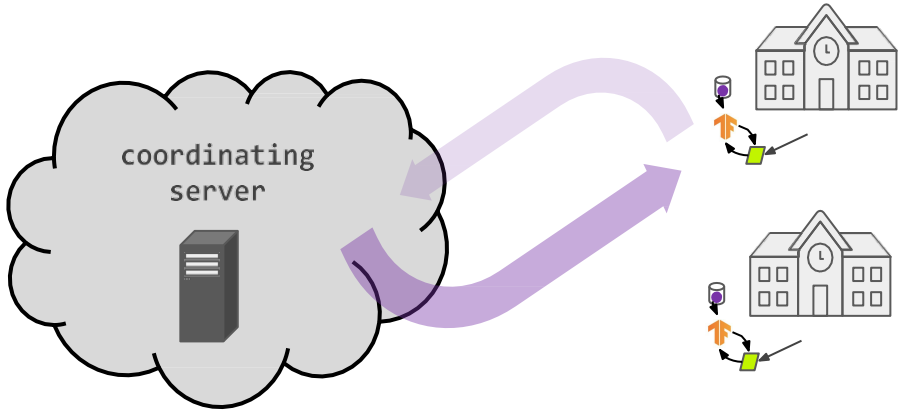
Cross-device federated learning

communication is often the primary bottleneck



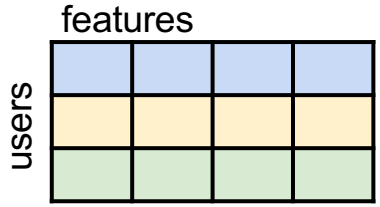
Cross-silo federated learning

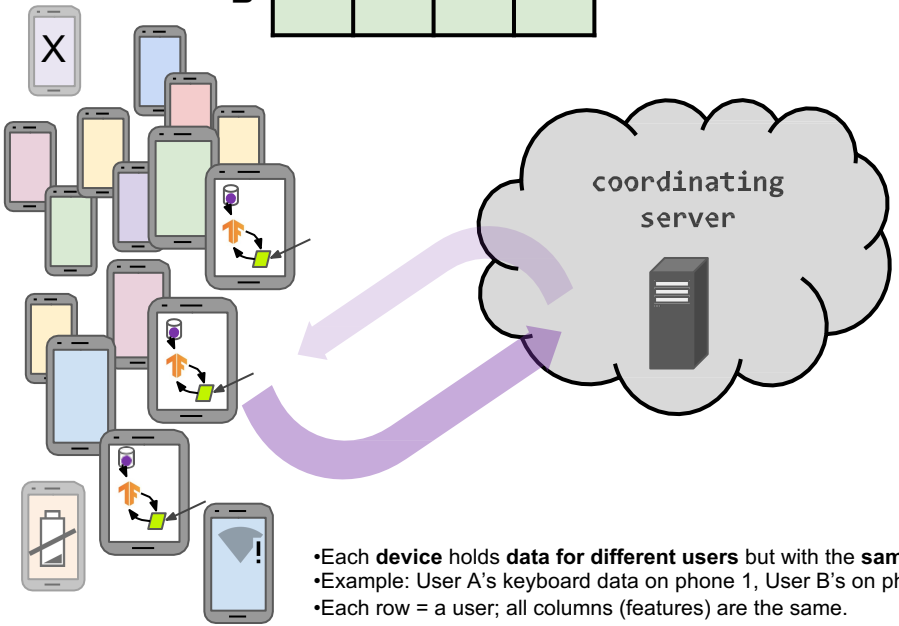
communication or computation might be the primary bottleneck



Cross-device federated learning

horizontally partitioned data

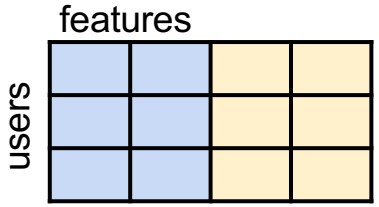


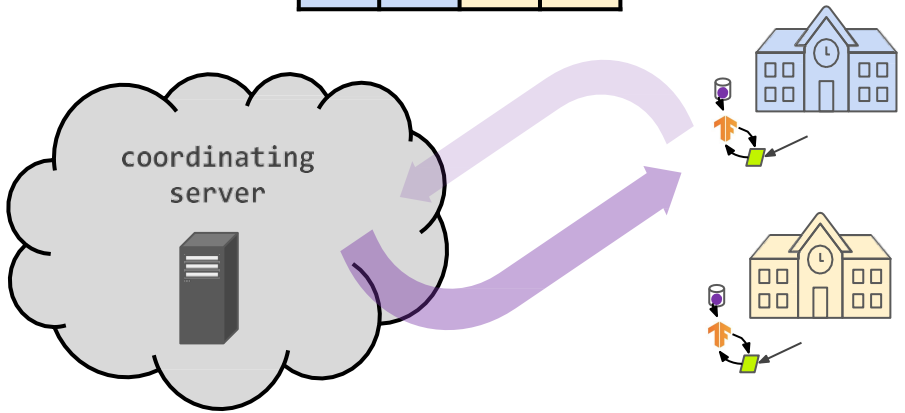


- Each **device** holds **data for different users** but with the **same features**.
- Example: User A's keyboard data on phone 1, User B's on phone 2.
- Each row = a user; all columns (features) are the same.

Cross-silo federated learning

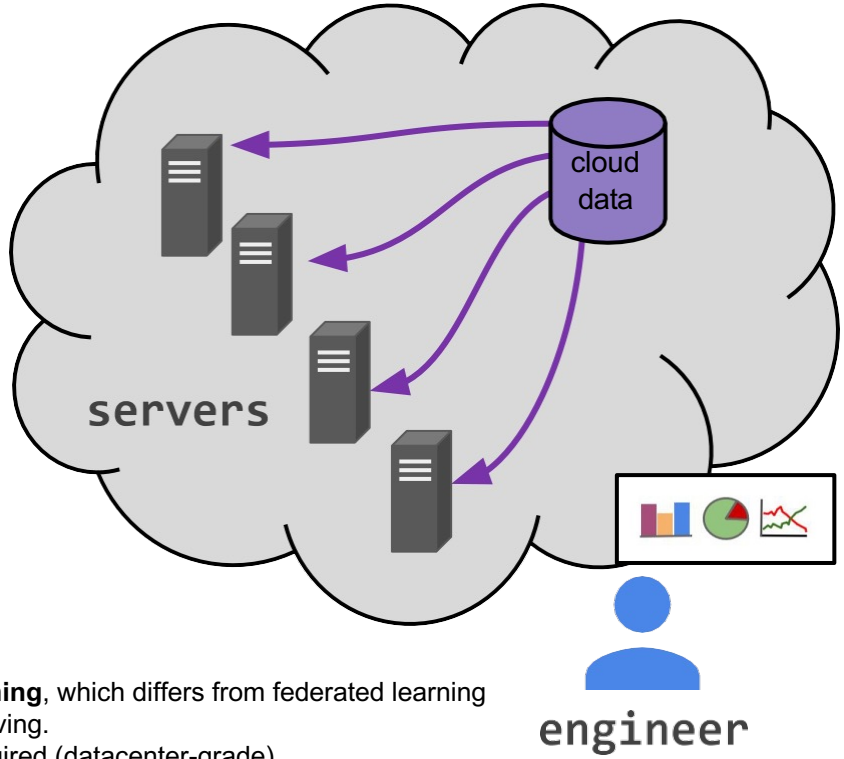
horizontal or
vertically partitioned data





- **Horizontal**: Like cross-device — different users, same features.
- **Vertical**: Same users, but **each silo holds different features**.
- Example: A bank holds financial data, a hospital holds health data for the same users.

Distributed datacenter machine learning



Distributed Datacenter Machine Learning, which differs from federated learning
- **Data is centralized**, not privacy-preserving.
- **High-performance infrastructure** required (datacenter-grade).
- Suitable for **large-scale training** (e.g., GPT, vision models).

FL terminology

- **Clients** - Compute nodes also holding local data, usually belonging to one entity:
  - IoT devices
  - Mobile devices
  - Data silos
  - Data centers in different geographic regions
- **Server** - Additional compute nodes that coordinate the FL process but don't access raw data. Usually not a single physical machine.

Characteristics of the federated learning setting

	Datacenter distributed learning	Cross-silo federated learning	Cross-device federated learning
Setting	Training a model on a large but "flat" dataset. Clients are compute nodes in a single cluster or datacenter.	Training a model on siloed data. Clients are different organizations (e.g., medical or financial) or datacenters in different geographical regions.	The clients are a very large number of mobile or IoT devices.
Data distribution	Data is centrally stored, so it can be shuffled and balanced across clients. Any client can read any part of the dataset.	**Data is generated locally and remains decentralized.** Each client stores its own data and cannot read the data of other clients. Data is not independently or identically distributed.	
Orchestration	Centrally orchestrated.	A central orchestration server/service organizes the training, but never sees raw data.	
Wide-area communication	None (fully connected clients in one datacenter/cluster).	Typically hub-and-spoke topology, with the hub representing a coordinating service provider (typically without data) and the spokes connecting to clients.	
Data availability	All clients are almost always available.		Only a fraction of clients are available at any one time, often with diurnal and other variations.
Distribution scale	Typically 1 - 1000 clients.	Typically 2 - 100 clients.	Massively parallel, up to 10^{10} clients.

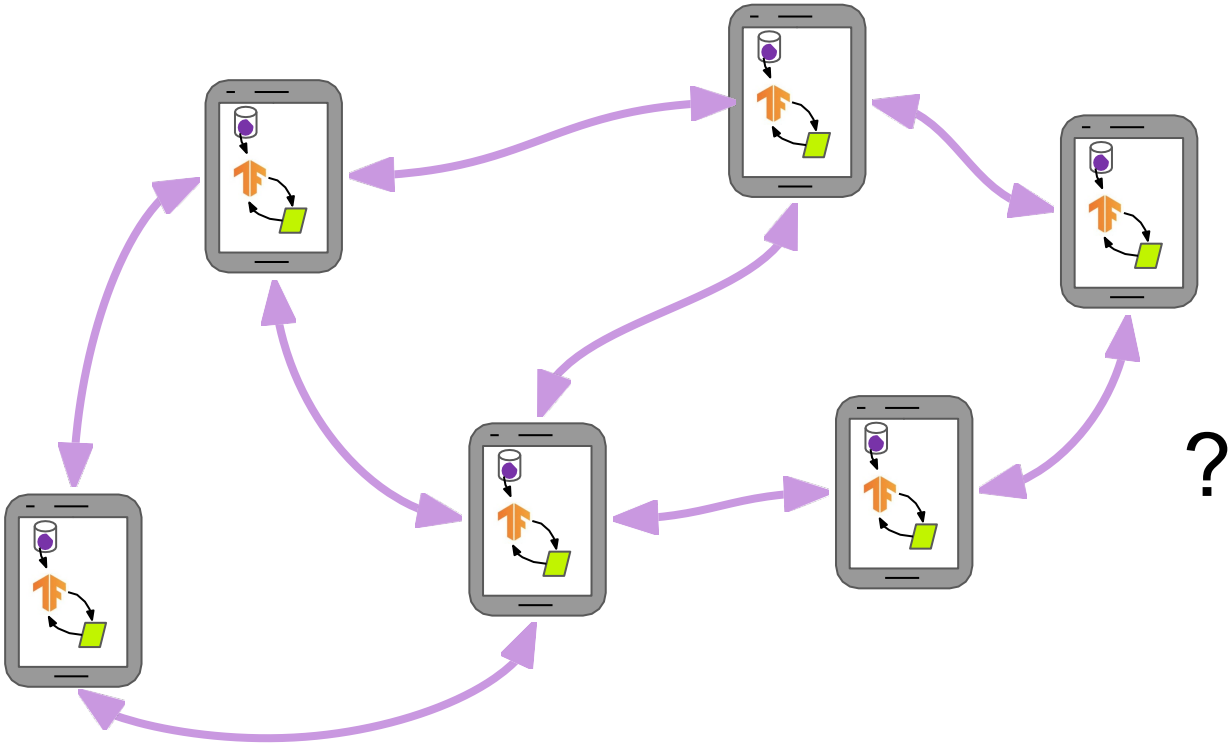
Adapted from Table 1 in *Advances and Open Problems in Federated Learning* (arxiv/1912.04977)

Characteristics of the federated learning setting

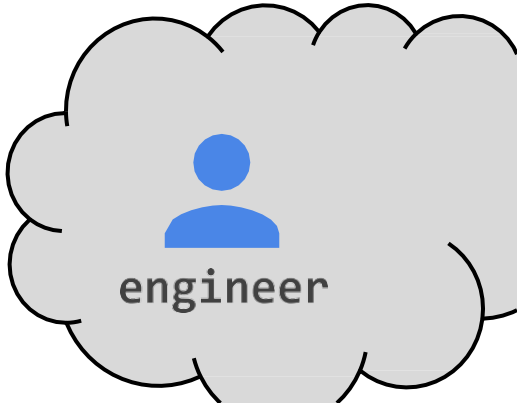
	Datacenter distributed learning	Cross-silo federated learning	Cross-device federated learning
Addressability	Each client has an identity or name that allows the system to access it specifically.		Clients cannot be indexed directly (i.e., no use of client identifiers)
Client statefulness	Stateful --- each client may participate in each round of the computation, carrying state from round to round.		Generally stateless --- each client will likely participate only once in a task, so generally we assume a fresh sample of never before seen clients in each round of computation.
Primary bottleneck	Computation is more often the bottleneck in the datacenter, where very fast networks can be assumed.	Might be computation or communication.	Communication is often the primary bottleneck, though it depends on the task. Generally, federated computations uses wi-fi or slower connections.
Reliability of clients	Relatively few failures.		Highly unreliable --- 5% or more of the clients participating in a round of computation are expected to fail or drop out (e.g., because the device becomes ineligible when battery, network, or idleness requirements for training/computation are violated).
Data partition axis	Data can be partitioned / re-partitioned arbitrarily across clients.	Partition is fixed. Could be example-partitioned (horizontal) or feature-partitioned (vertical).	Fixed partitioning by example (horizontal).

Adapted from Table 1 in *Advances and Open Problems in Federated Learning* (arxiv/1912.04977)

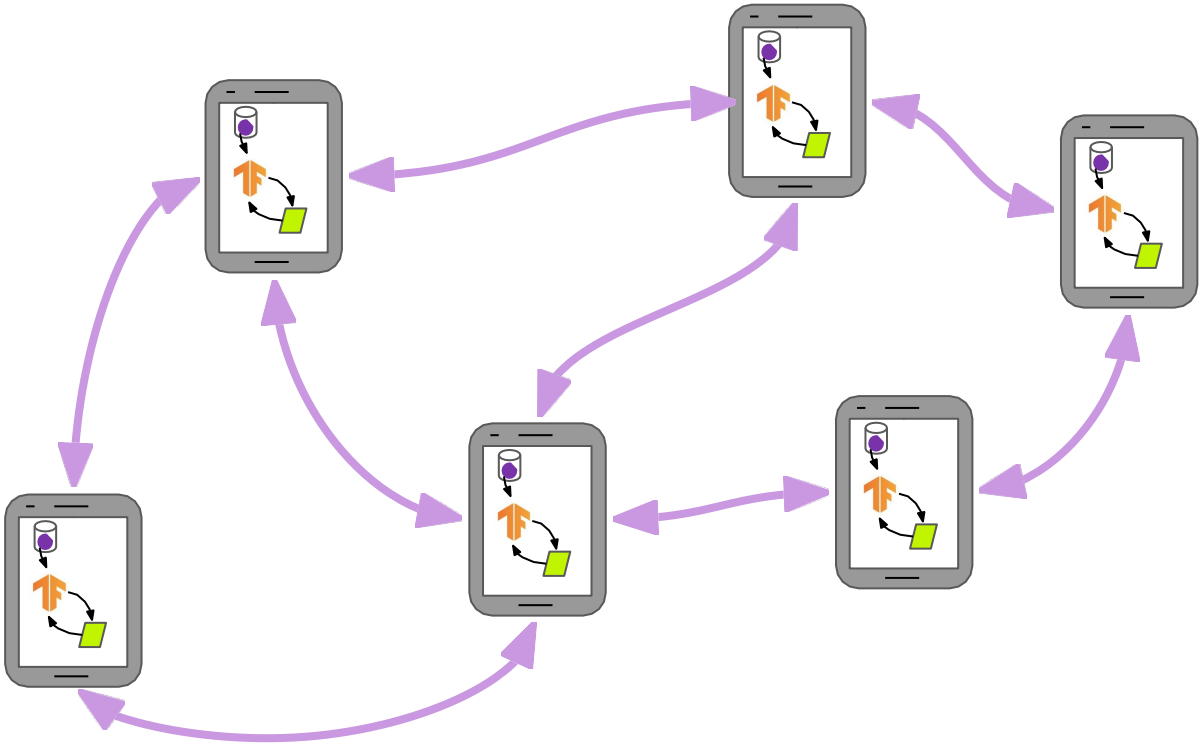
Fully decentralized (peer-to-peer) learning



?



Fully decentralized (peer-to-peer) learning



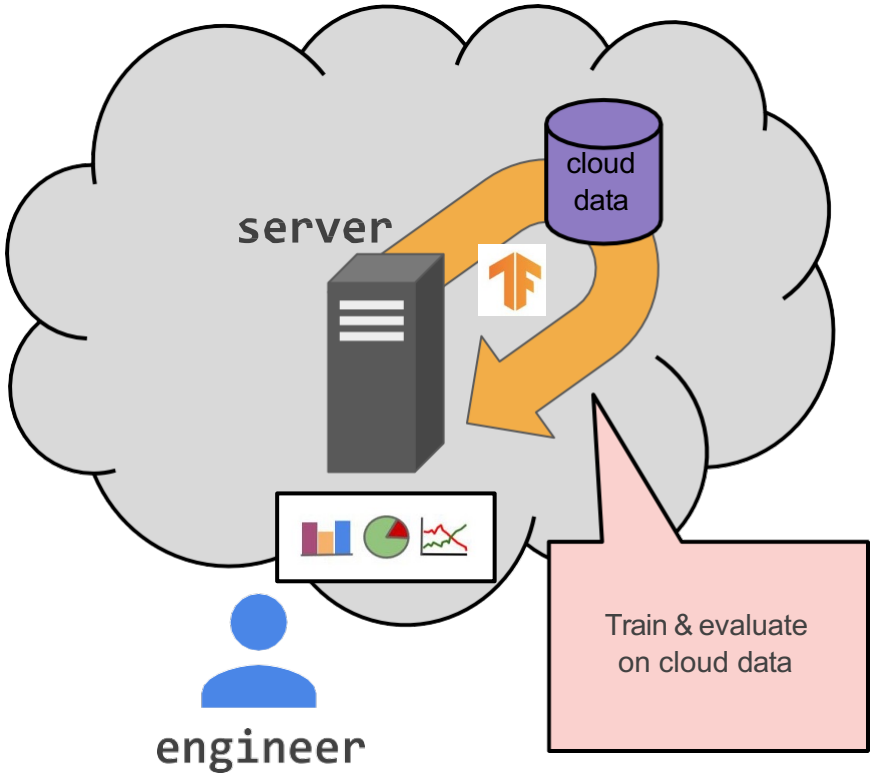
Characteristics of FL vs decentralized learning

	Federated learning	**Fully decentralized (peer-to-peer) learning**
Orchestration	A central orchestration server/service organizes the training, but never sees raw data.	No centralized orchestration.
Wide-area communication pattern	Typically hub-and-spoke topology, with the hub representing a coordinating service provider (typically without data) and the spokes connecting to clients.	Peer-to-peer topology.

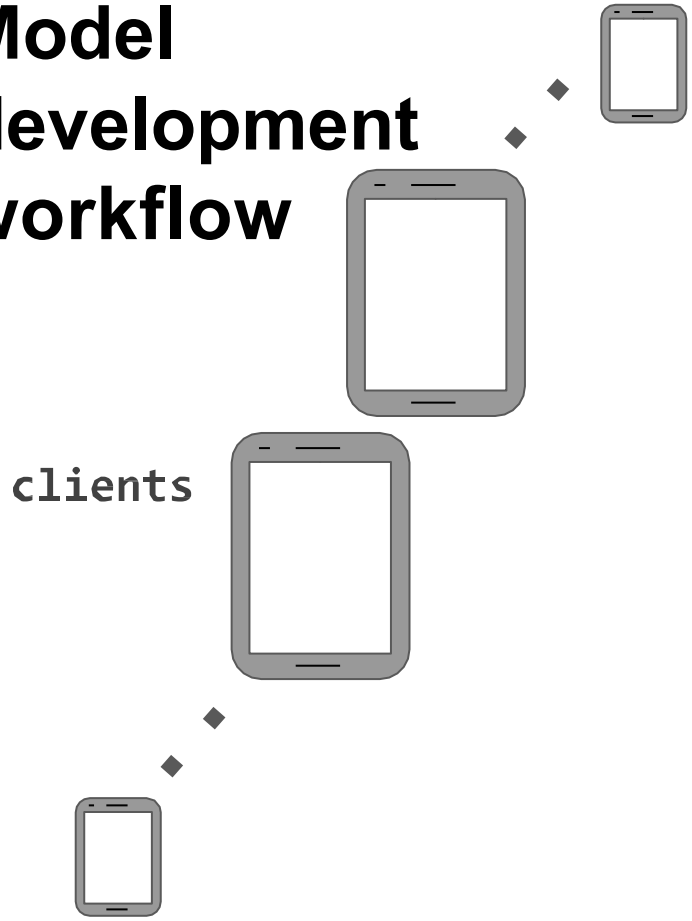
Adapted from Table 3 in *Advances and Open Problems in Federated Learning* (arxiv/1912.04977)

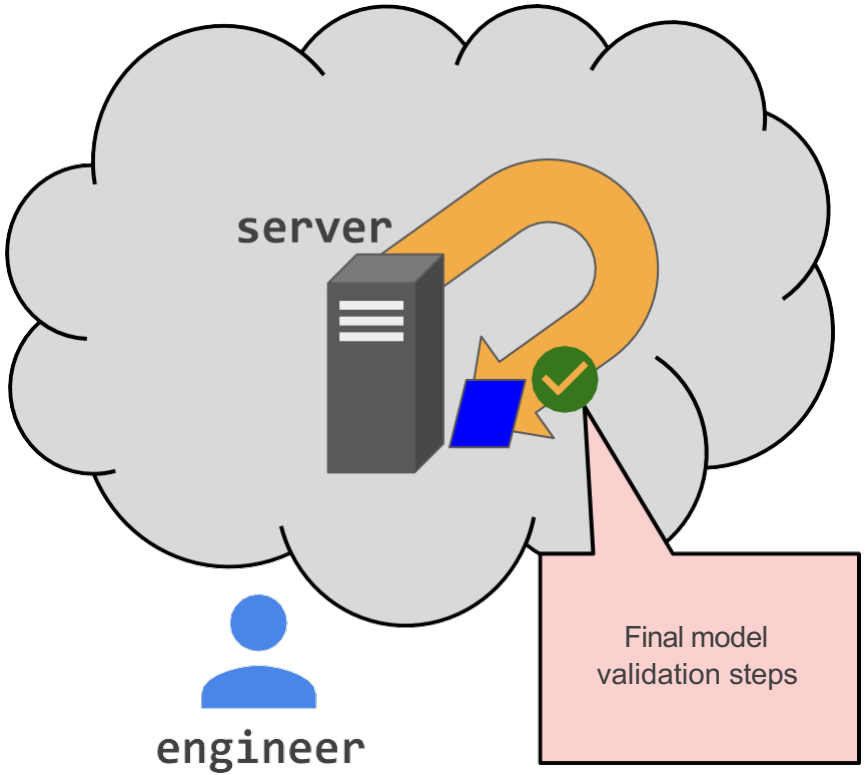
Cross-Device Federated Learning

Model development workflow

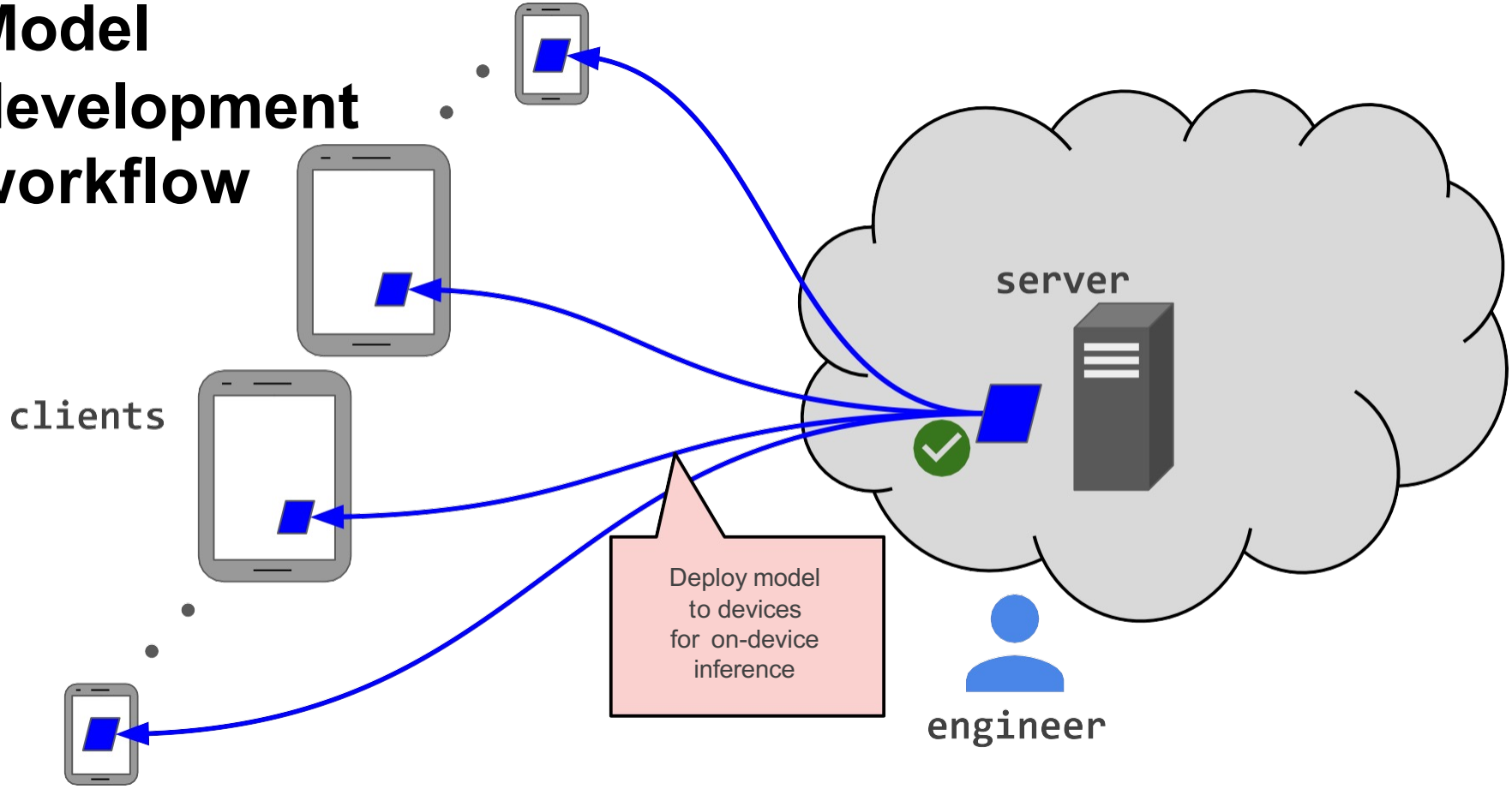


Model development workflow

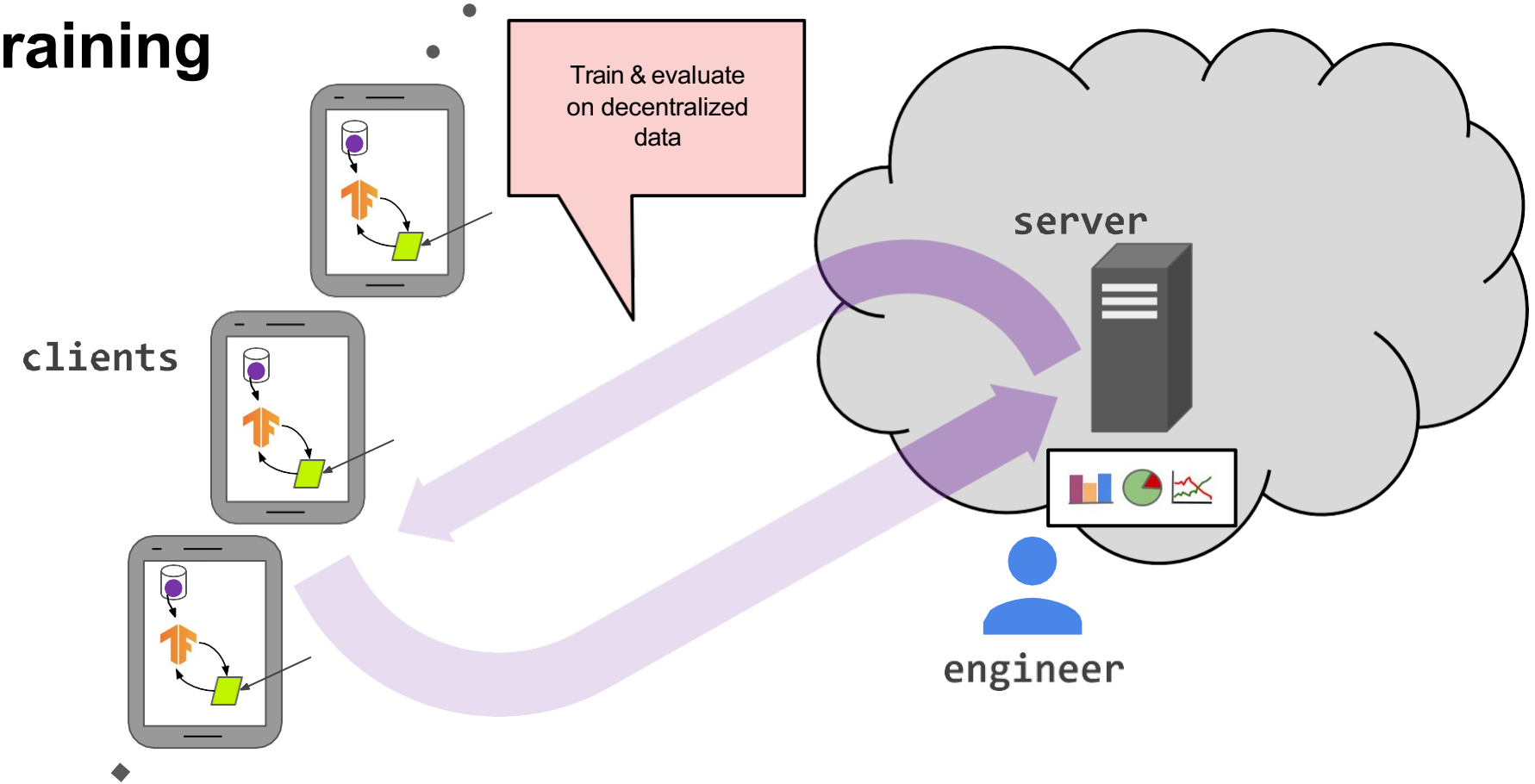




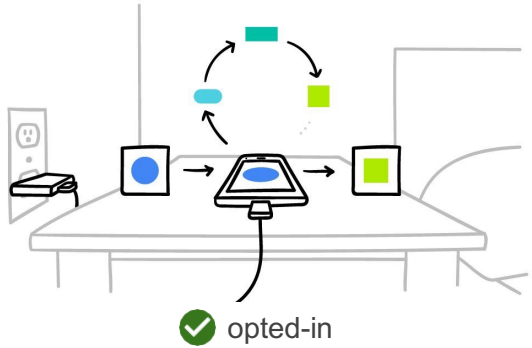
Model development workflow

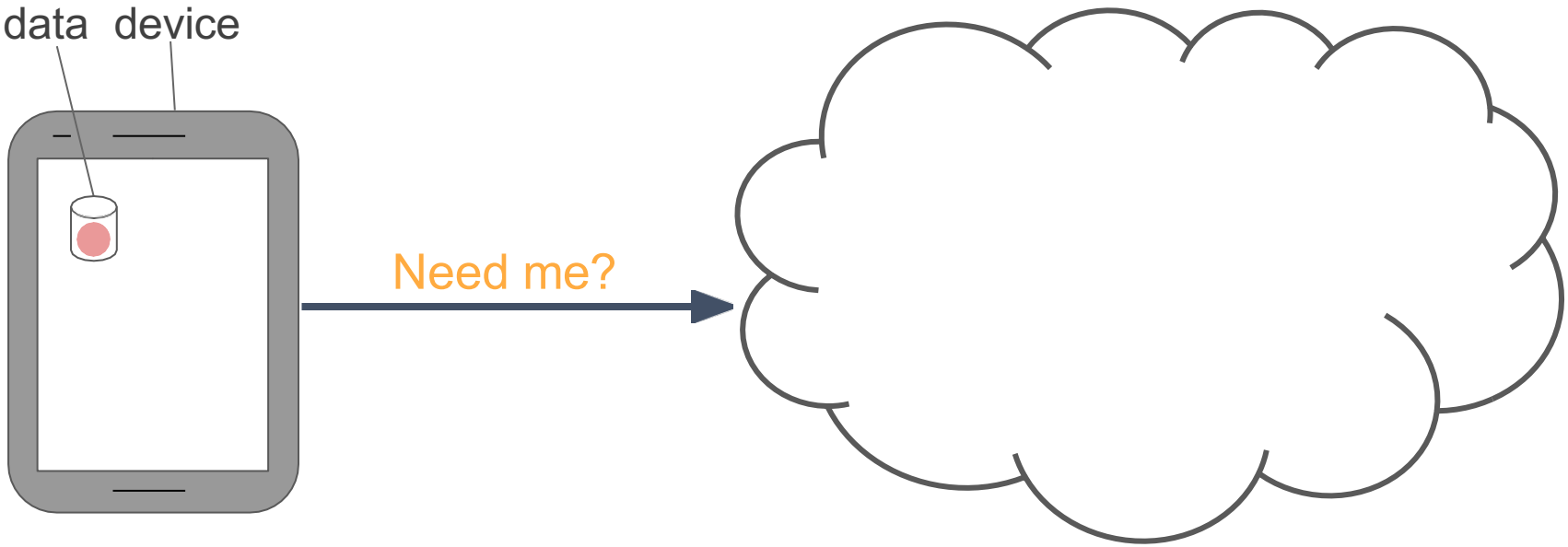


Federated training

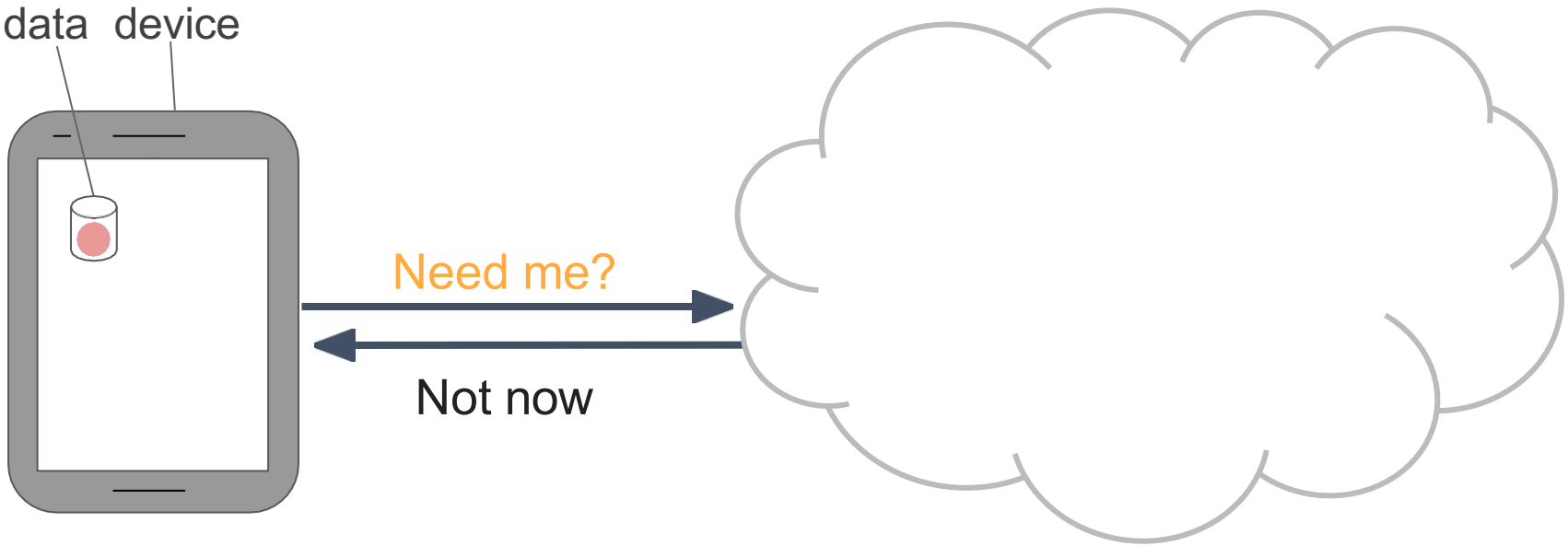


Federated learning

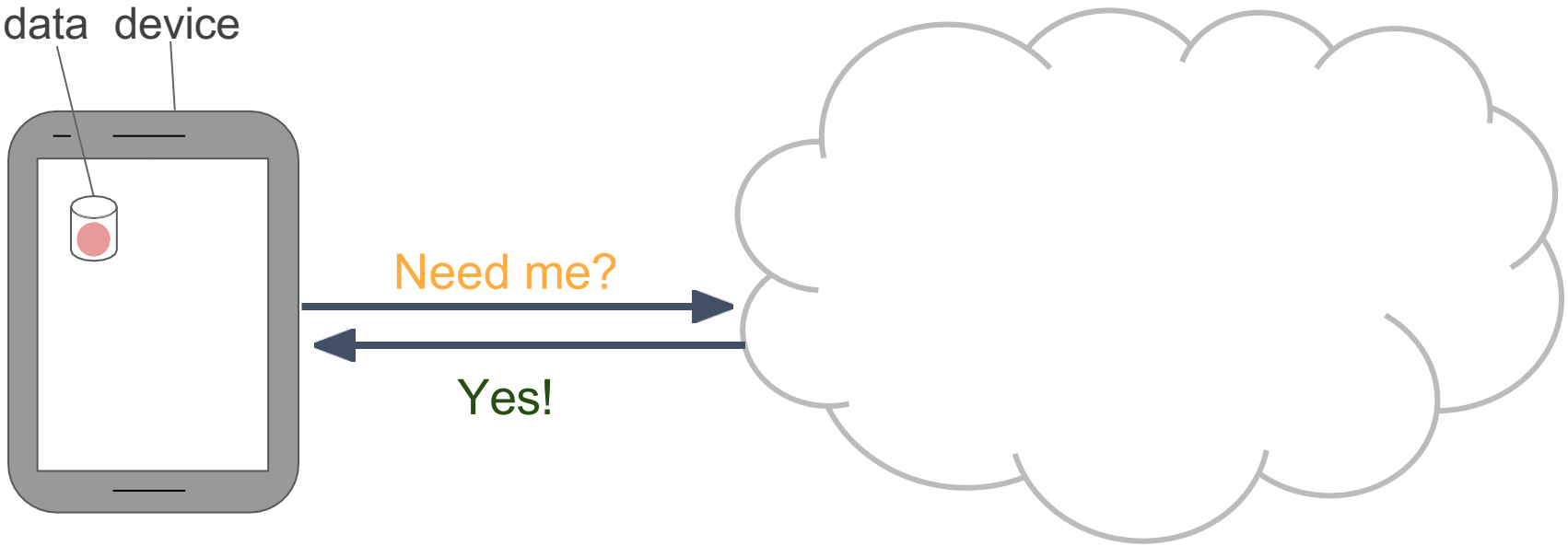




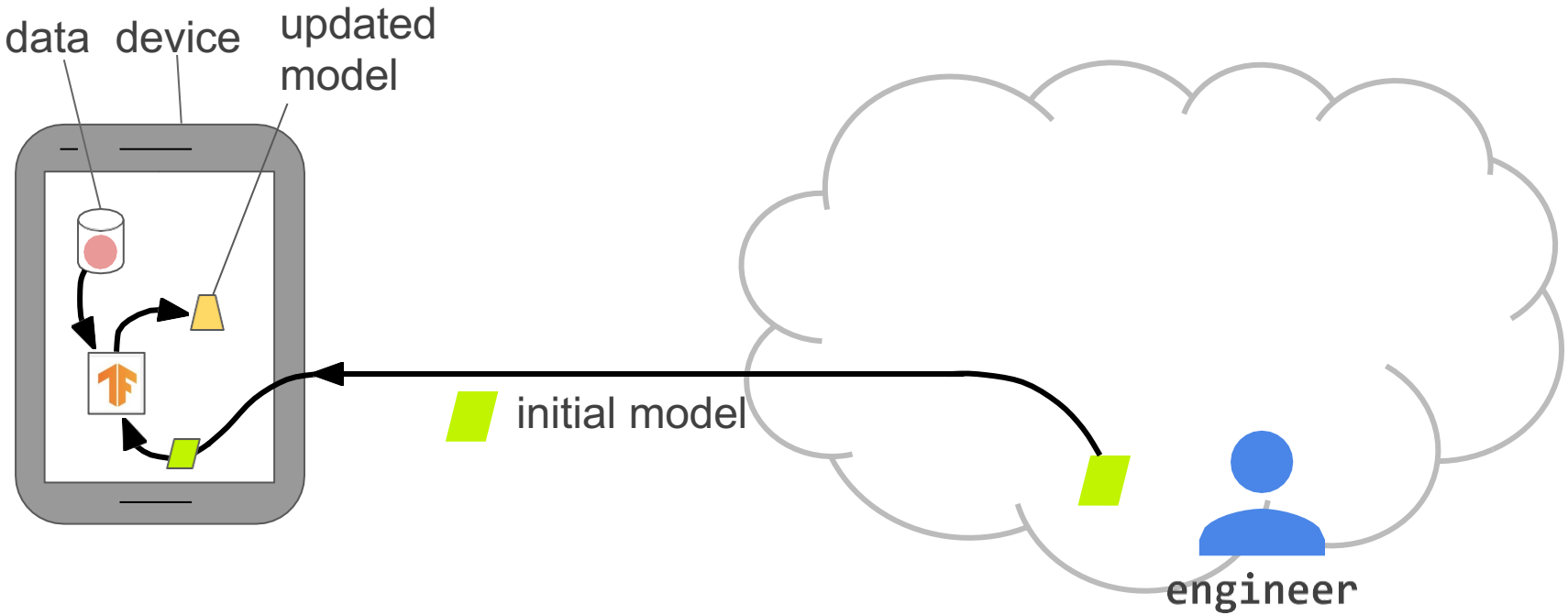
Federated learning



Federated learning

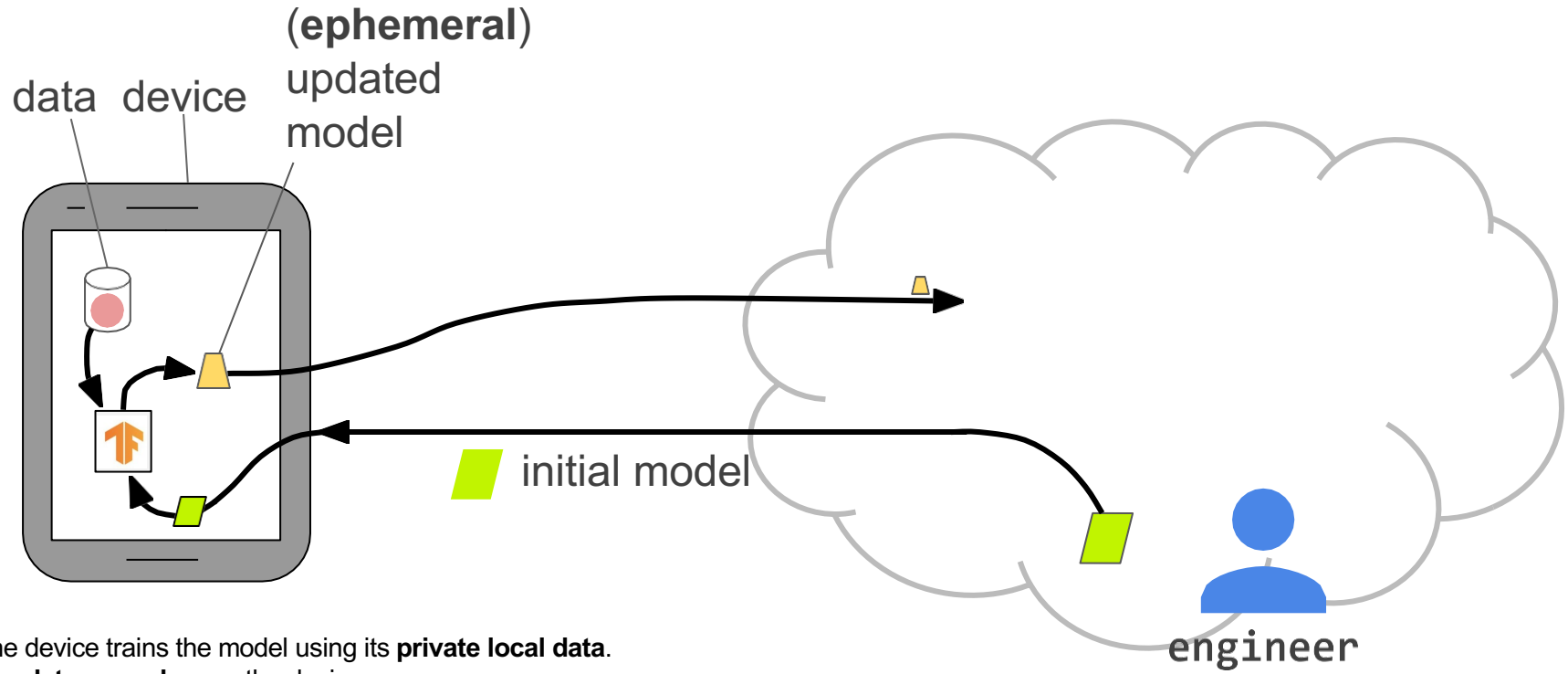


Federated learning



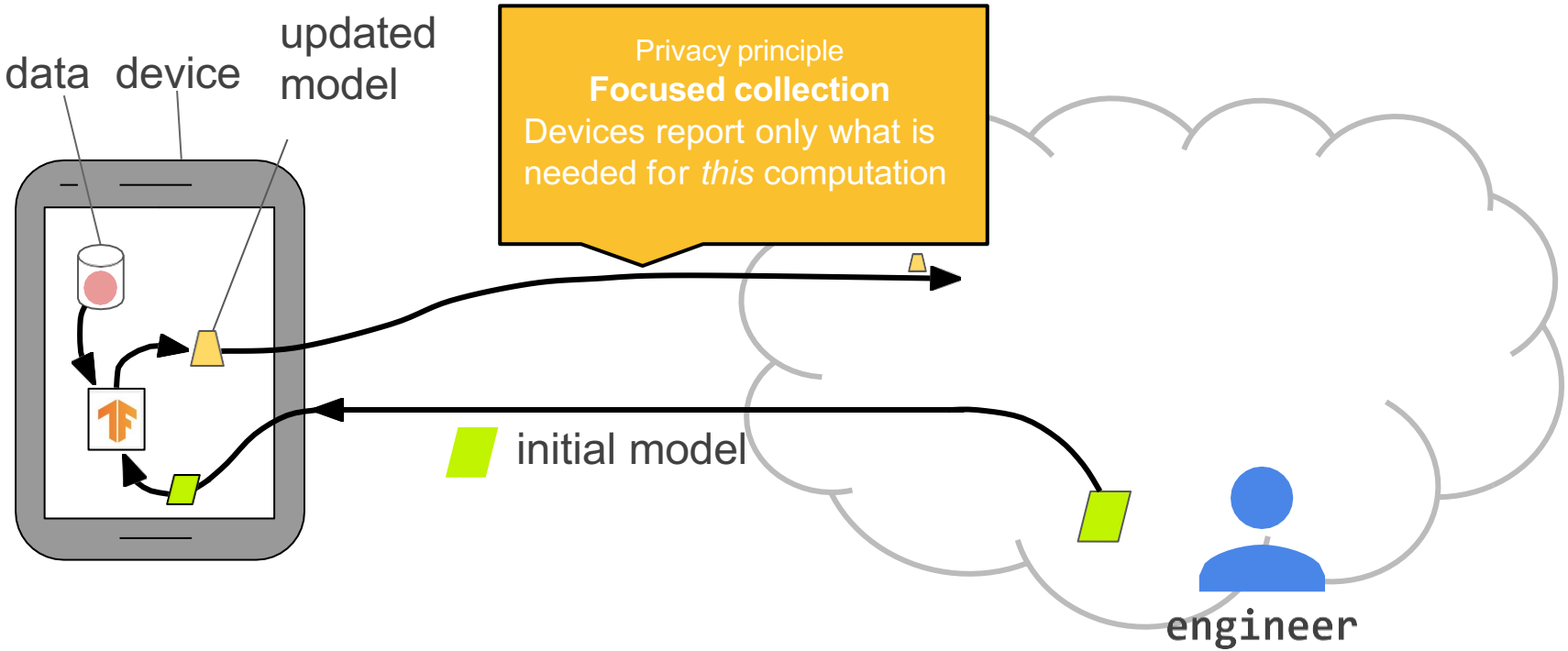
Federated learning

The device produces an **updated model (or model gradient)**.
Once the update is sent, it would be deleted



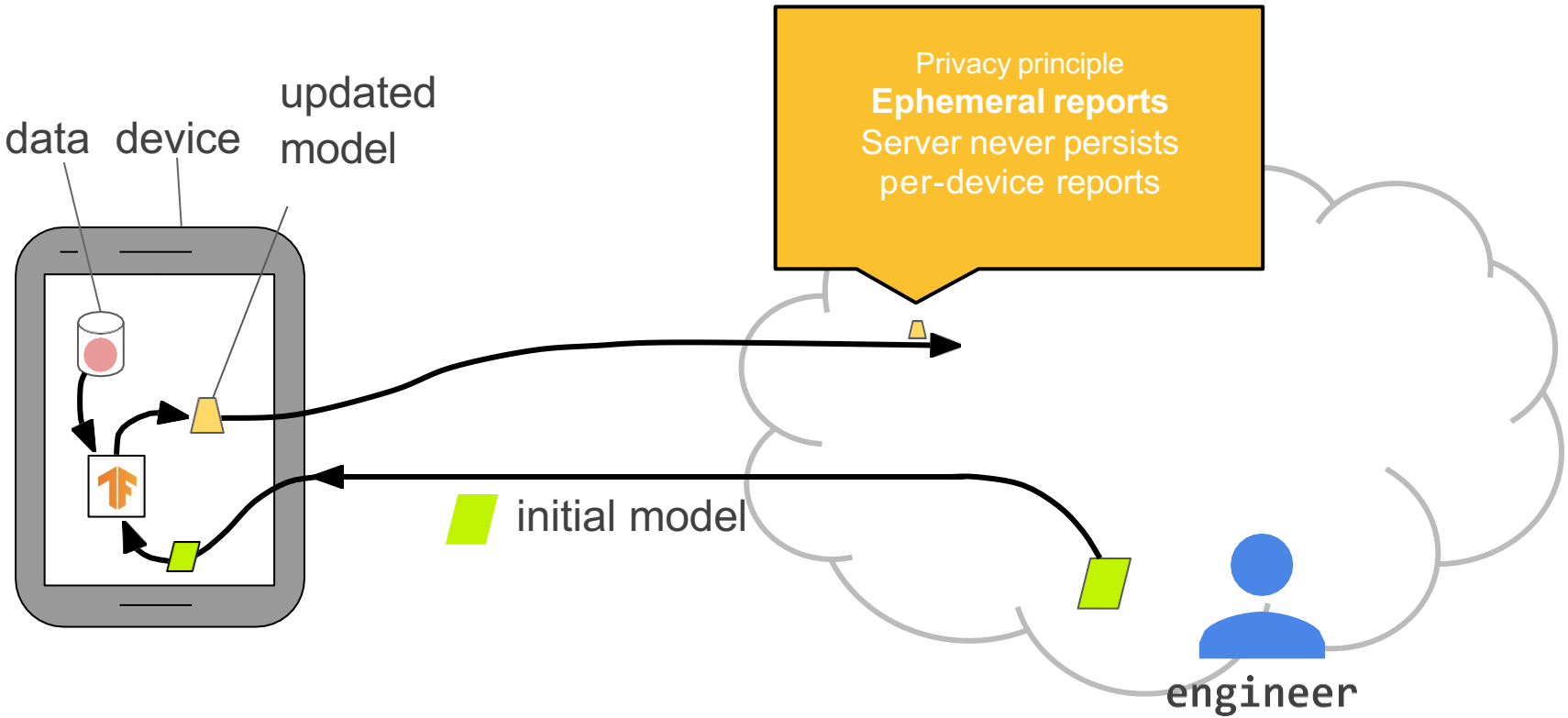
The device trains the model using its **private local data**.
The **data never leaves** the device.

Federated learning

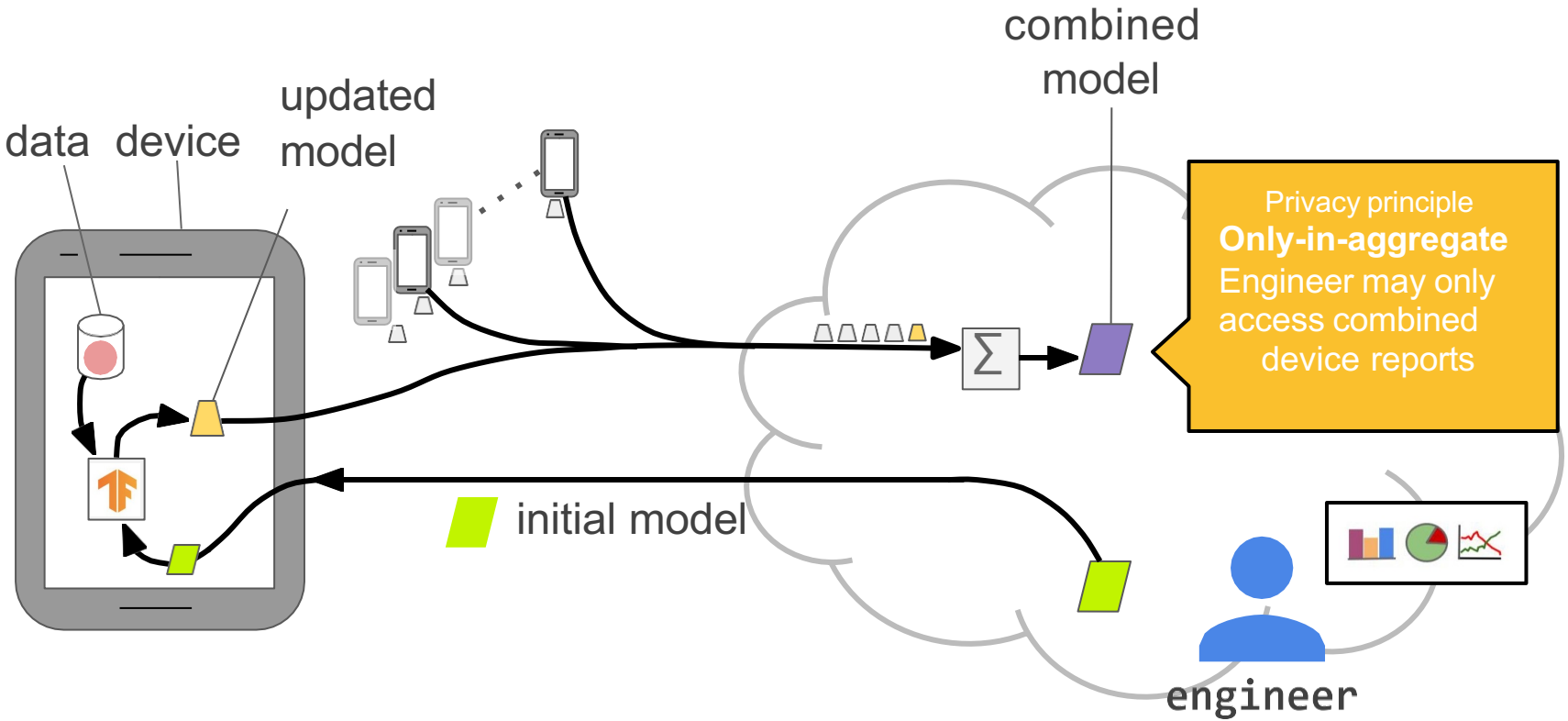


The engineer **never sees raw data** — just aggregated results.

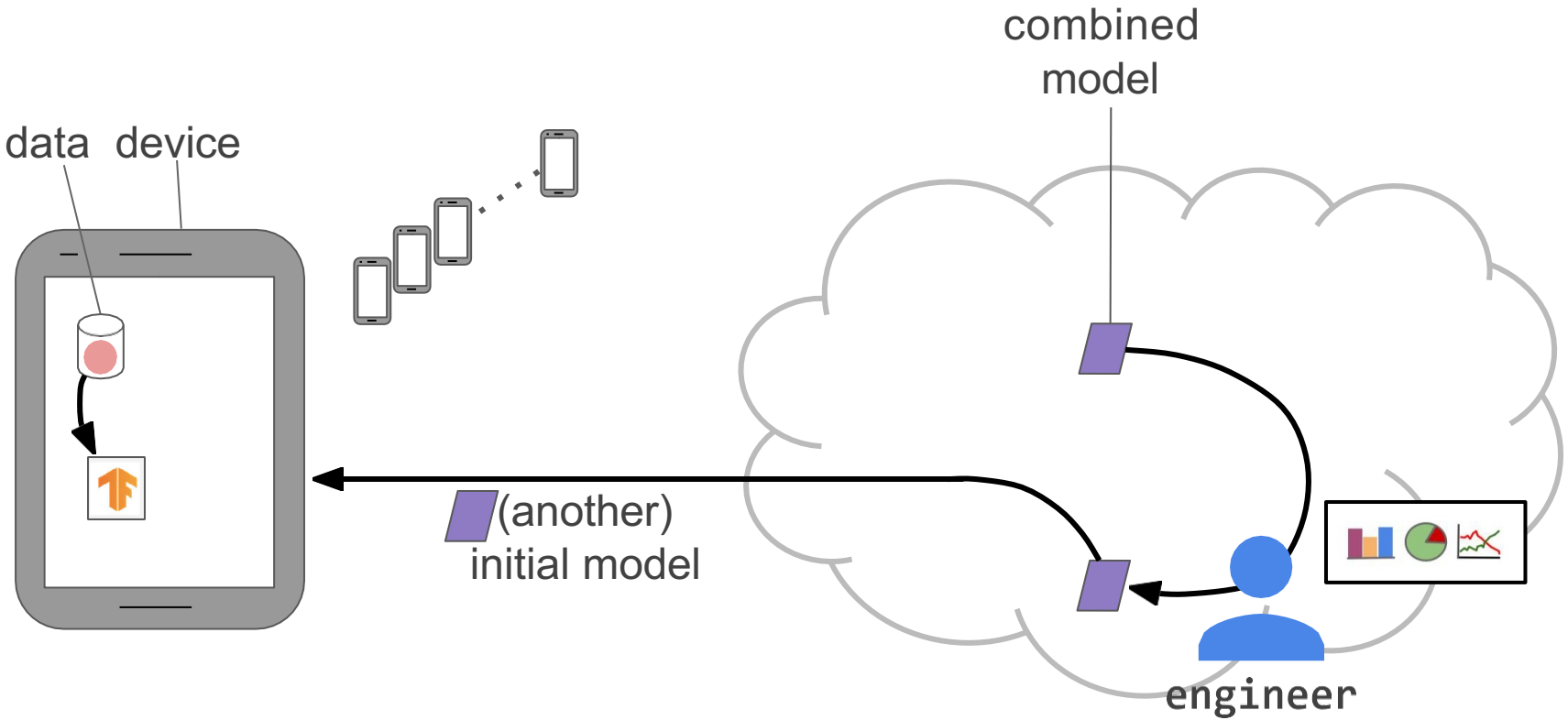
Federated learning



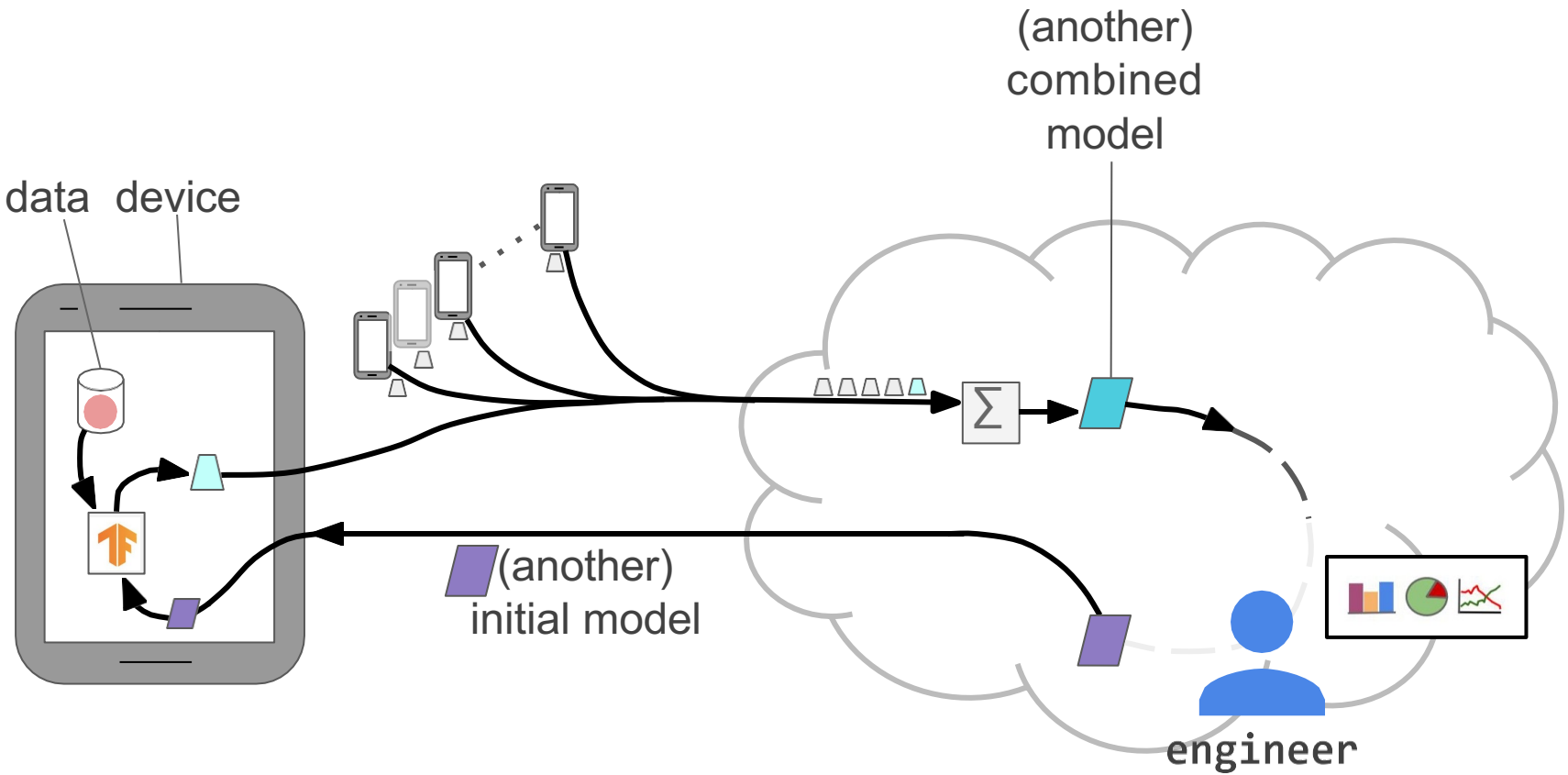
Federated learning



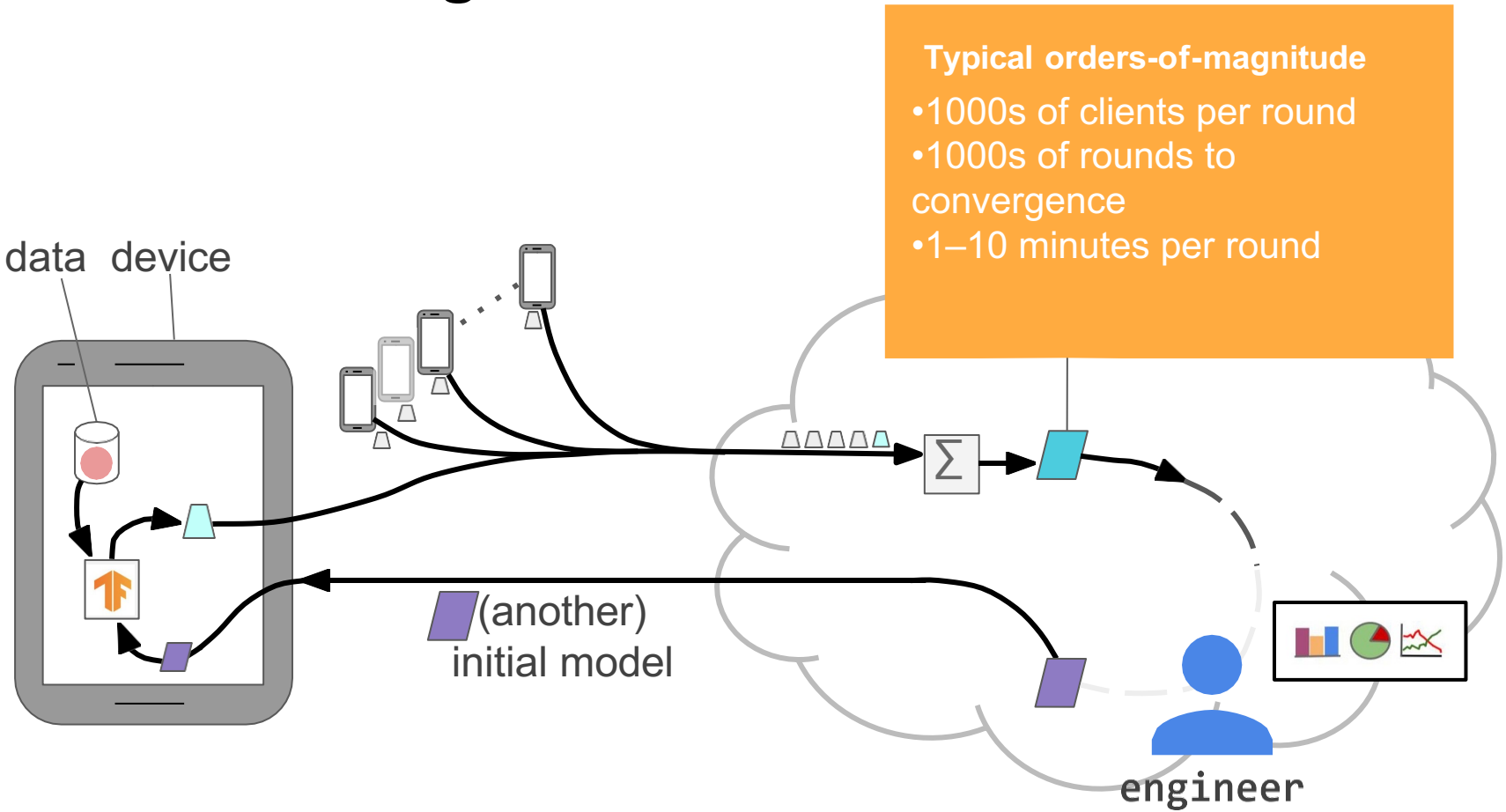
Federated learning



Federated learning

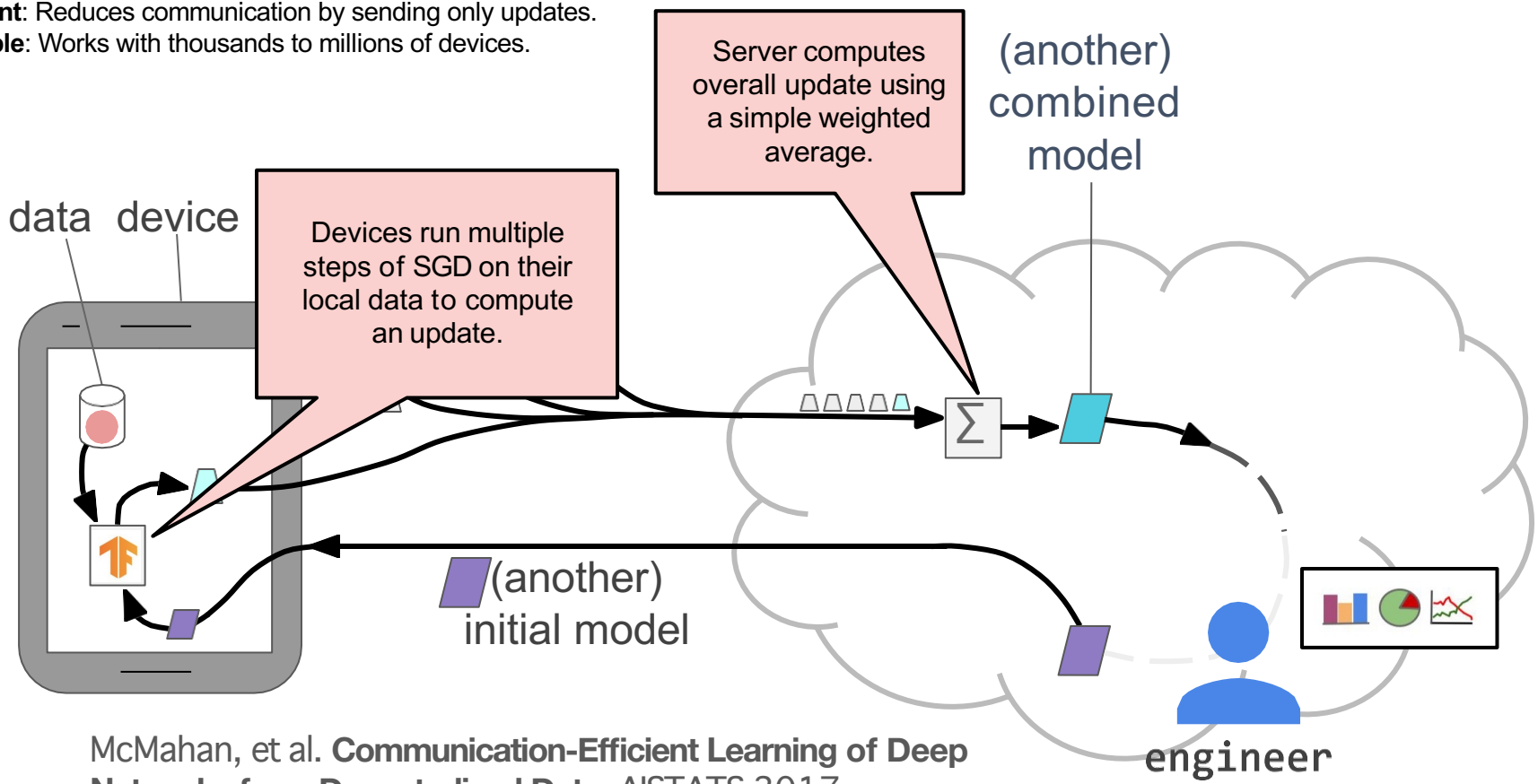


Federated learning



Federated Averaging (FedAvg) algorithm

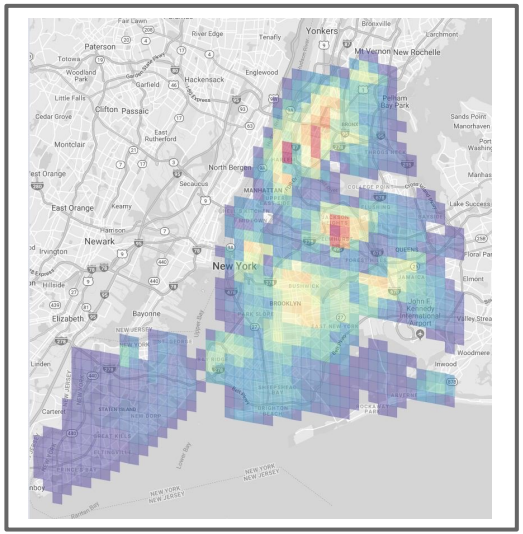
Efficient: Reduces communication by sending only updates.
Scalable: Works with thousands to millions of devices.



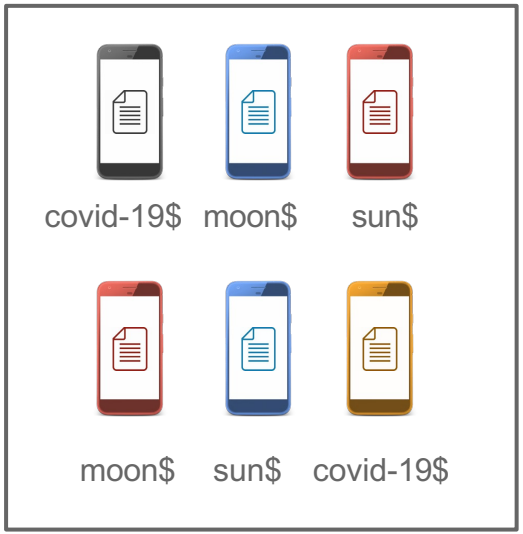
McMahan, et al. **Communication-Efficient Learning of Deep Networks from Decentralized Data.** AISTATS 2017.

Beyond Learning: Federated Analytics

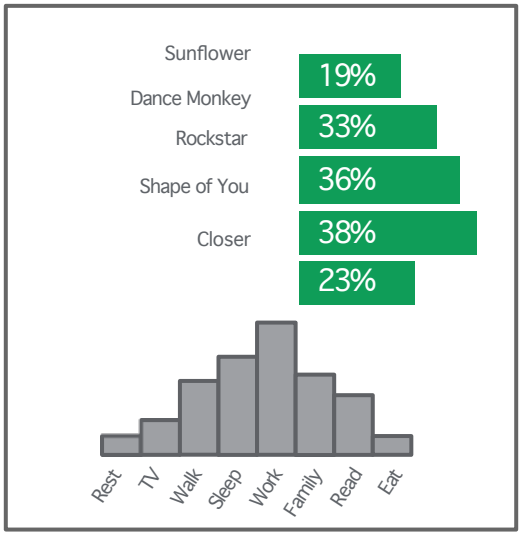
Beyond learning: federated analytics



Geo-location heatmaps



Frequently typed
out-of-dictionary words



Popular songs, trends,
and activities

Federated Analytics uses **on-device computation** and **aggregated insights** without collecting raw data.

Federated analytics

Federated analytics is the practice of applying data science methods to the analysis of raw data that is stored locally on users' devices. Like federated learning, it works by running local computations over each device's data, and only making the aggregated results — and never any data from a particular device — available to product engineers. Unlike federated learning, however, federated analytics aims to support basic data science needs.

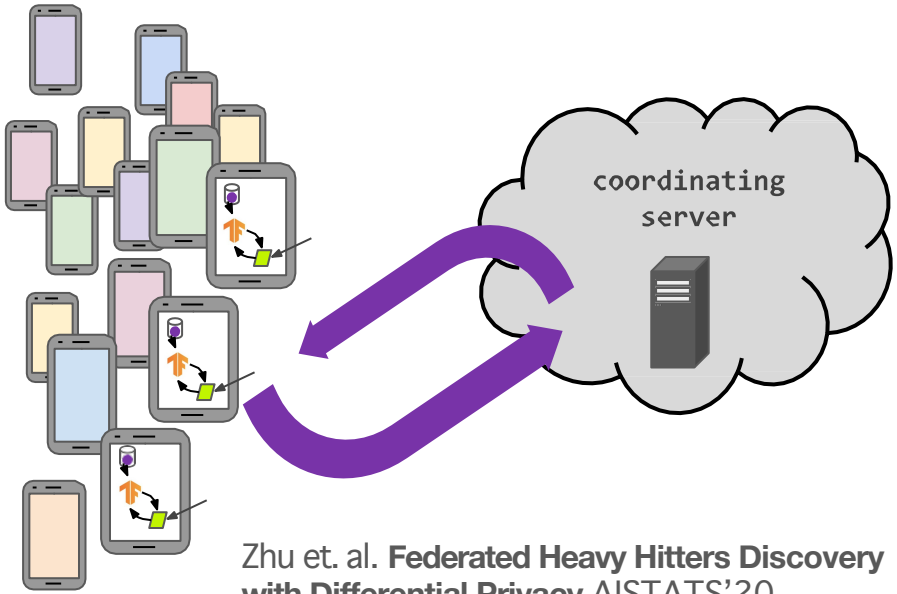
definition proposed in https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html

Federated analytics

- Federated histograms over closed sets
- Federated quantiles and distinct element counts
- Federated heavy hitters discovery over open sets
- Federated density of vector spaces
- Federated selection of random data subsets
- Federated SQL
- Federated computations?
- etc...

Interactive algorithms

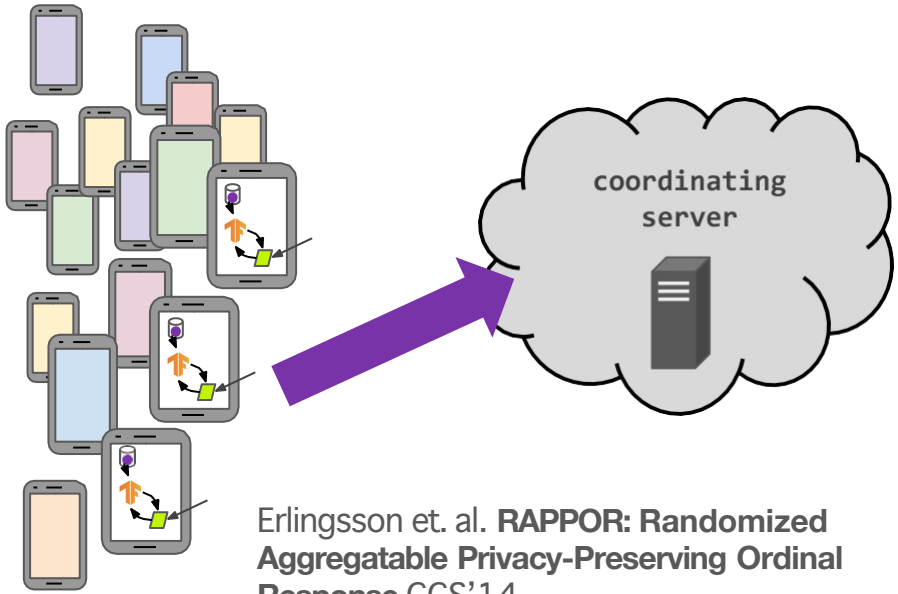
Similar to learning, the on-device computation is a function of a server state



Zhu et. al. **Federated Heavy Hitters Discovery with Differential Privacy** AISTATS'20.

Non-interactive algorithms

Unlike learning, the on-device computation does not depend on a server state



Erlingsson et. al. **RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response** CCS'14.

Tutorial Outline

Part 1: What is Federated Learning?

Part 2: Federated Optimization

Part 3: Privacy for Federated Technologies

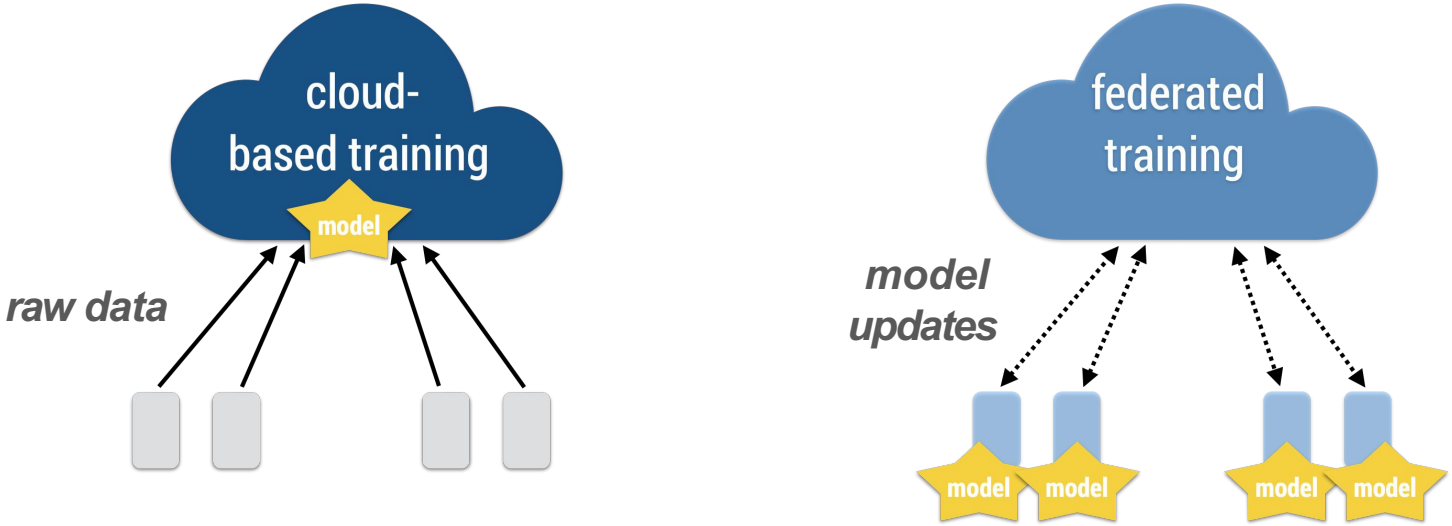
Part 4: Open Problems and Other Topics

Part II: Federated Optimization

Federated optimization

Train ML models at the edge (on-device or on-site), rather than sending data to the cloud.

goal: train machine learning models at the edge



why? ✓ reduce strain on network ✓ privacy ✓ quickly incorporate new data

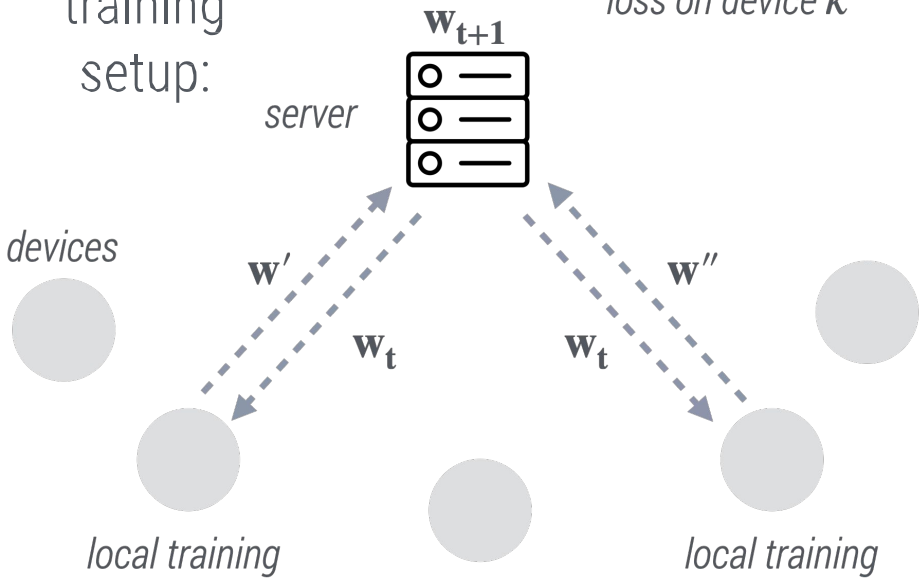
Federated optimization: workflow & challenges

p: Weight of device k

objective: $\min_w f(w) = \sum_{k=1}^{m} p_k F_k(w)$

loss on device k

training setup:



w_t:
The **global model** at round t, shared by the server.

w', w'':
The **locally updated models** after devices train on their private data.

w_{t+1}:
The **new global model**, obtained by **aggregating** the local models.

Challenges

expensive communication
- massive, slow networks

privacy concerns
- user privacy constraints

statistical heterogeneity
- unbalanced, non-IID data

systems heterogeneity
- variable hardware, connectivity, etc

Goal: Minimize **global loss** across all devices without moving data

A CLOSER LOOK

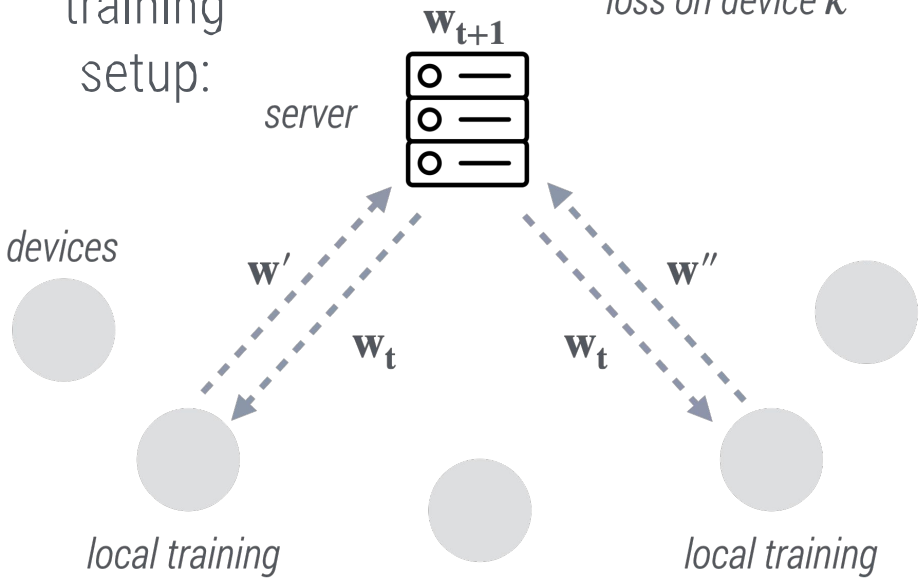
Federated optimization: workflow

ERM: Empirical Risk Minimization

objective: $\min_w f(w) = \sum_{k=1}^{m} p_k F_k(w)$

loss on device k

training setup:



Typically consider solving an ERM objective, which is a (possibly) weighted average of losses across the *m* devices and their local data, i.e.,

$$\min_w \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$$

However, challenges discussed translate to other common ML objectives as well

n_k: number of data samples on device k

ℓ: loss function (e.g. cross-entropy)

$h(x; w)$: model's prediction

$(x_k^{(i)}, y_k^{(i)})$: data sample i on device k

ERM objective

Risk

$$R(h) = \mathbb{E}_{(x,y)\sim P}[\ell(h(x; w), y)]$$

Assume we have access to a sample of data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$

Goal is to estimate the expected risk using this sample, i.e.:

Empirical Risk

$$R_{emp}(h) = \frac{1}{N}\sum_{i=1}^{N} \ell(h(x^{(i)}; w), y^{(i)})$$

Federated ERM objective

$h(x; w)$: Model's prediction

ℓ: Loss function (e.g. cross-entropy)

P_k: Data distribution on device k

Q: Distribution over devices (used to sample clients)

Risk

$$R(h) = \mathbb{E}_{k \sim Q} \mathbb{E}_{(x,y) \sim P_k} [\ell(h(x; w), y)]$$

We have a sample of data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ **generated from *m* devices**

Each device may generate data according to its own distribution, P_k

Goal is to estimate the true risk over this sample, i.e.:

Empirical Risk

$$R_{emp}(h) = \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$$

Terminology

m	number of devices
N	total number of data points
n_k	number of data points on device k
$(x_k^{(i)}, y_k^{(i)})$	i-th data point on device k
w	model parameters

Empirical Risk

$$R_{emp}(h) = \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$$
$$= \sum_{k=1}^{m} p_k F_k(w)$$

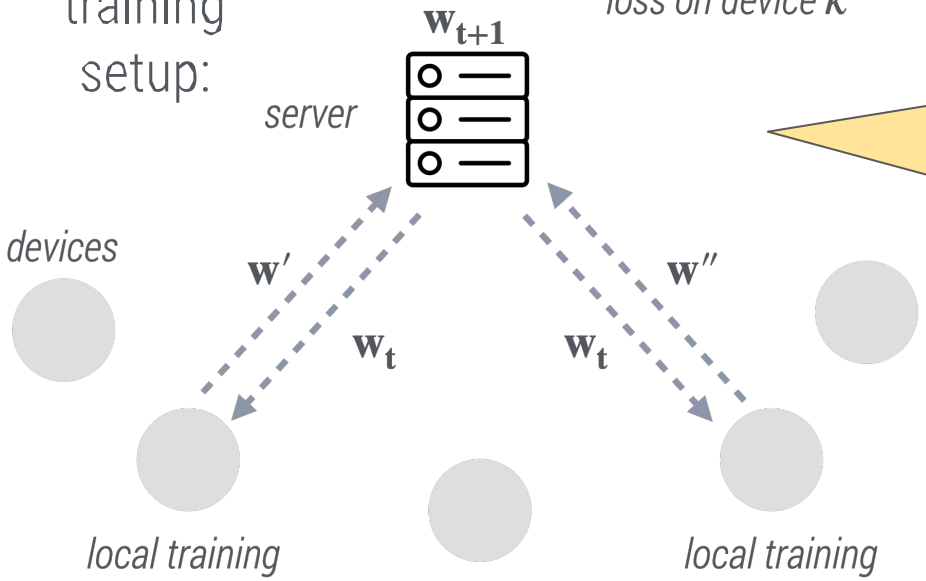
A CLOSER LOOK

Federated optimization: workflow

objective: $\min_w f(w) = \sum_{k=1}^{m} p_k F_k(w)$

F_k: loss on device k

training setup:



Here we consider the problem of ***cross-device*** federated learning, and assume a centralized server

Will discuss relaxations and alternatives in Part IV

A CLOSER LOOK

Federated optimization: challenges

Can **reduce communication** in federated optimization by:

1. Limiting *number of devices* involved in communication
2. Reducing number of *communication rounds*
3. Reducing *size of messages* sent over network

expensive communication

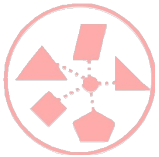
- massive, slow networks

privacy concerns

- user privacy constraints

statistical heterogeneity

- unbalanced, non-IID data



systems heterogeneity

- variable hardware, connectivity, etc

A CLOSER LOOK

Federated optimization: challenges

Keeping **raw data local** to each device is a first step

Will discuss privacy mechanisms that can be used in conjunction with federated optimization in Part III

expensive communication

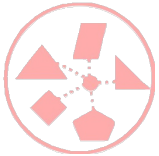
- massive, slow networks

privacy concerns

- user privacy constraints

statistical heterogeneity

- unbalanced, non-IID data



systems heterogeneity

- variable hardware, connectivity, etc

A CLOSER LOOK

Federated optimization: challenges

- Each device's data comes from a **different distribution**
- Example: One user types in English, another in Spanish
- This breaks the IID (independent and identically distributed) assumption of traditional ML

Heterogeneous (i.e., non-identically distributed) data and systems can bias optimization procedures

expensive communication

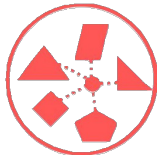
- massive, slow networks

privacy concerns

- user privacy constraints

statistical heterogeneity

- unbalanced, non-IID data



systems heterogeneity

- variable hardware, connectivity, etc

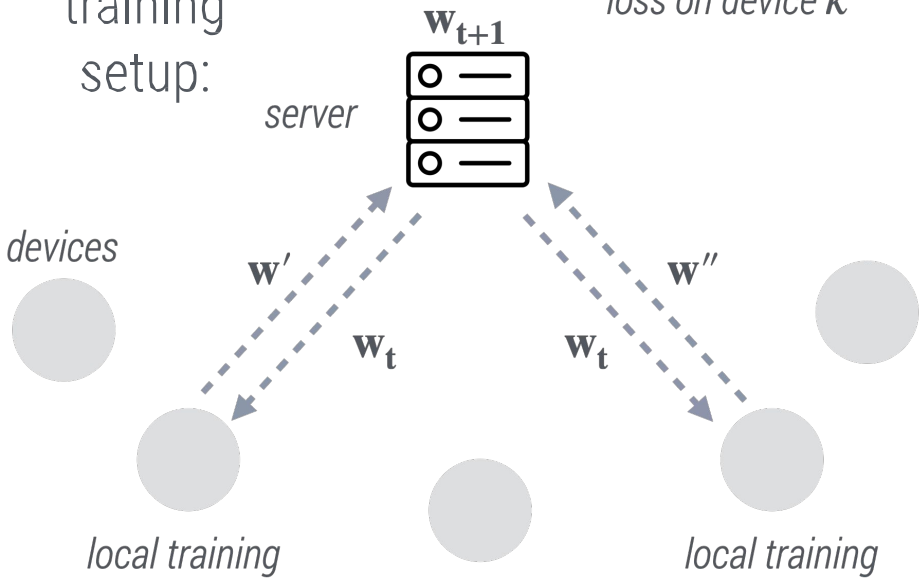
Devices have **different compute, memory, battery, and uptime**

Federated optimization: workflow & challenges

objective: $\min_w f(w) = \sum_{k=1}^{m} p_k F_k(w)$

loss on device k

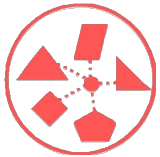
training setup:



expensive communication
- massive, slow networks

privacy concerns
- user privacy constraints

statistical heterogeneity
- unbalanced, non-IID data



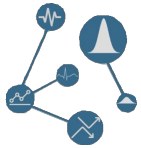
systems heterogeneity
- variable hardware, connectivity, etc

Federated optimization: workflow & challenges

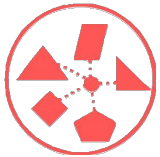
Federated Learning: Challenges, Methods, and Future Directions,
T. Li, A. K. Sahu, A. Talwalkar, V. Smith,
IEEE Signal Processing Magazine 2020

expensive communication
- massive, slow networks

privacy concerns
- user privacy constraints



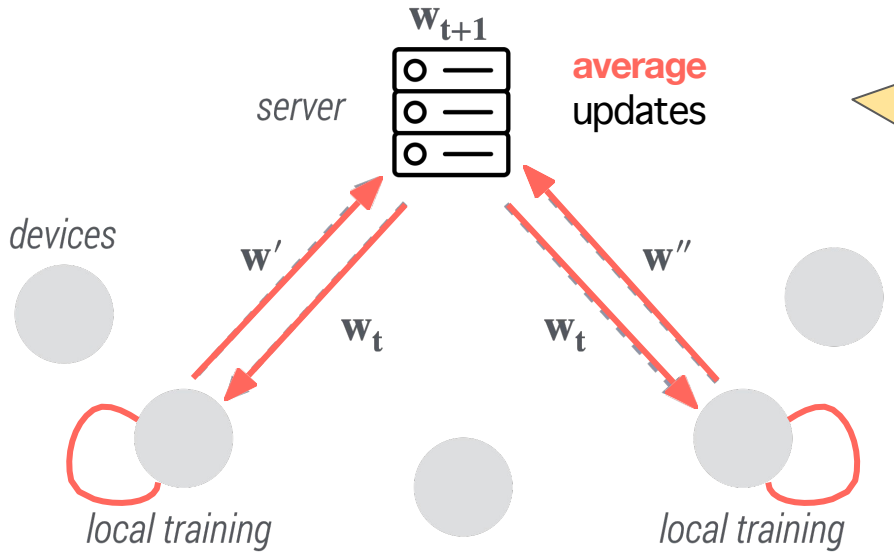
statistical heterogeneity
- unbalanced, non-IID data



systems heterogeneity
- variable hardware, connectivity, etc

A STANDARD BASELINE

Federated Averaging (FedAvg)



- At each communication round:
  (i) run SGD locally, then
  (ii) average the model updates
- Can add privacy mechanisms to procedure (more later …)
- Reduces communication by:
  (i) performing local updating,
  (ii) communicating with a subset of devices

•Clients do multiple local steps → fewer server round-trips.
•Often only a **subset of devices** participate each round.

How does FedAvg differ from distributed SGD?

Each device processes **just one mini-batch**
Does one local update per communication round:

Distributed SGD: computation on device k

for $i \in$ *mini-batch* B
  $\Delta \mathbf{w} \leftarrow \Delta \mathbf{w} - \alpha \nabla f_i(\mathbf{w})$
end
$\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$

Each device performs **T** local SGD steps

FedAvg: computation on device k

for $t = 1, 2, \ldots,$ *local iterations* T
  $\Delta \mathbf{w} \leftarrow \Delta \mathbf{w} - \alpha \nabla f_{i_t}(\mathbf{w})$
  $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
end

Why is it useful to perform `local-updating`?

1. Can perform more local computation (i.e., more than just one mini-batch)
2. Incorporate updates more quickly (immediately apply gradient information)

✓ **Can lead to method converging in many fewer communication rounds**

How does FedAvg differ from distributed SGD?

MNIST CNN, 99% ACCURACY

CNN	E	B	u	IID		Non-IID	
FedSGD	1	∞	1	626		483	
FedAvg	5	∞	5	179	(3.5×)	1000	(0.5×)
FedAvg	1	50	12	65	(9.6×)	600	(0.8×)
FedAvg	20	∞	20	234	(2.7×)	672	(0.7×)
FedAvg	1	10	60	34	(18.4×)	350	(1.4×)
FedAvg	5	50	60	29	(21.6×)	334	(1.4×)
FedAvg	20	50	240	32	(19.6×)	426	(1.1×)
FedAvg	5	10	300	20	(31.3×)	229	(2.1×)
FedAvg	20	10	1200	18	(34.8×)	173	(2.8×)

SHAKESPEARE LSTM, 54% ACCURACY

LSTM	E	B	u	IID		Non-IID	
FedSGD	1	∞	1.0	2488		3906	
FedAvg	1	50	1.5	1635	(1.5×)	549	(7.1×)
FedAvg	5	∞	5.0	613	(4.1×)	597	(6.5×)
FedAvg	1	10	7.4	460	(5.4×)	164	(23.8×)
FedAvg	5	50	7.4	401	(6.2×)	152	(25.7×)
FedAvg	5	10	37.1	192	(13.0×)	41	(95.3×)

Local-updating (FedAvg) can reduce communication rounds by ~100x relative to SGD

E: Number of local epochs

B: Local batch size

u: Number of clients sampled per round

IID vs Non-IID: Balanced vs skewed data across clients

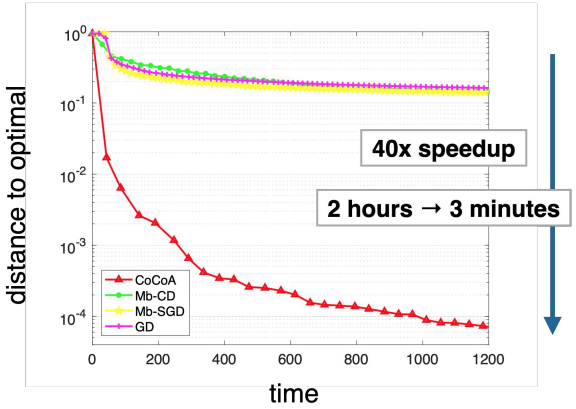
Lower values in the IID/Non-IID columns = **fewer communication rounds** needed to reach the target accuracy

[McMahan et al, Communication-efficient learning of deep networks from decentralized data, AISTATS 2017]

ASIDE

Local-updating is not new

- Extreme setting: one-shot averaging
  (e.g., [Zhang, Duchi, Wainwright, Communication-Efficient Algorithms for Statistical Optimization, JMLR 2013])
- Consensus-based optimization
  - ADMM
    [Boyd et al, Distributed Optimization and Statistical Learning via ADMM, FnT in ML, 2010]
  - CoCoA
    [Jaggi & Smith et al, Communication-Efficient Distributed Dual Coordinate Ascent, NeurIPS 2014]



ASIDE

Local-updating is not new

- Extreme setting: one-shot averaging
  (e.g., [Zhang, Duchi, Wainwright, Communication-Efficient Algorithms for Statistical Optimization, JMLR 2013])
- Consensus-based optimization
  - ADMM
    [Boyd et al, Distributed Optimization and Statistical Learning via ADMM, FnT in ML, 2010]
  - CoCoA
    [Jaggi & Smith et al, Communication-Efficient Distributed Dual Coordinate Ascent, NeurIPS 2014]
- Local-SGD
  (e.g., [MacDonald et al, Efficient large-scale distributed training of conditional maxent models, NeurIPS 2009])
- Decentralized optimization

Federated setting is distinct in considering local-updating with heterogeneous data and partial device participation, often for non-convex objectives

while local updating is old, **federated learning is new in its constraints and use cases**.

How does FedAvg differ from distributed SGD?

Distributed SGD: computation on device k

for $i \in$ *mini-batch* B
  $\Delta \mathbf{w} \leftarrow \Delta \mathbf{w} - \alpha \nabla f_i(\mathbf{w})$
end
$\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$

FedAvg: computation on device k

for $t = 1, 2, \ldots,$ *local iterations* T
  $\Delta \mathbf{w} \leftarrow \Delta \mathbf{w} - \alpha \nabla f_{i_t}(\mathbf{w})$
  $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
end

Why is it useful to perform `local-updating`?

1. Can perform more local computation (i.e., more than just one mini-batch)
2. Incorporate updates more quickly (immediately apply gradient information)

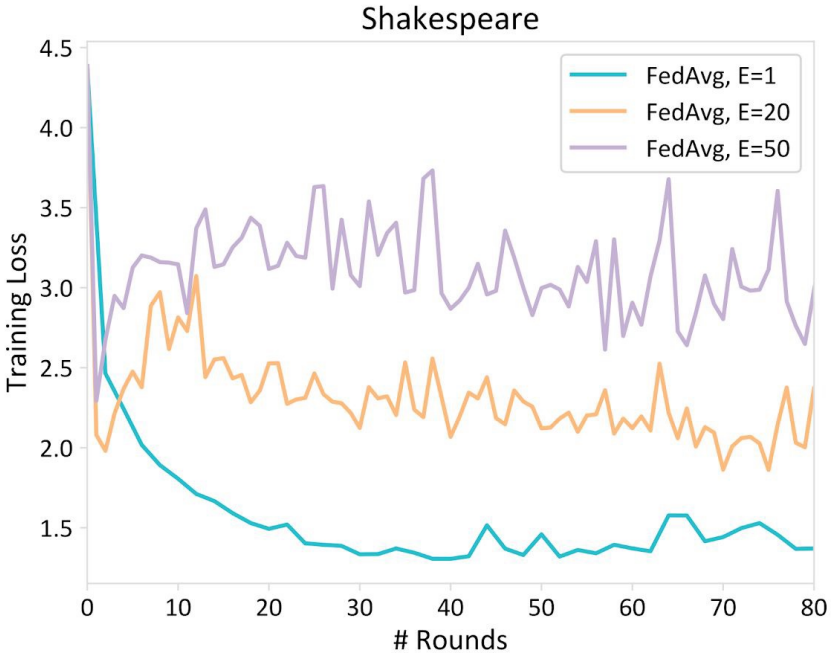
✓ **Can lead to method converging in many fewer communication rounds**

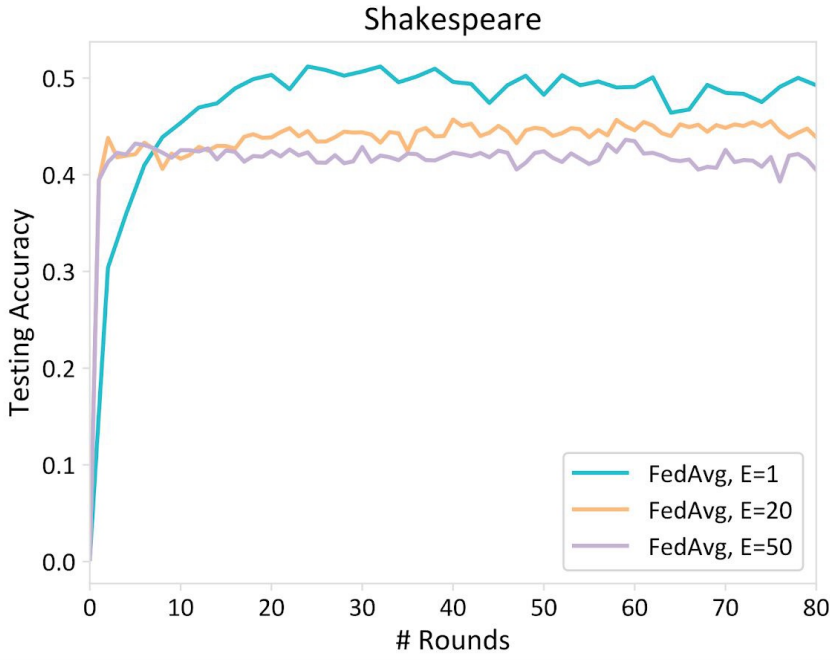
✗ **But, can potentially hurt convergence if not properly tuned …**

WILL THIS CONVERGE?

Challenge: heterogeneity

E: number of local epochs





There's a **trade-off**:

- **Small E** = better accuracy, more communication
- **Large E** = fewer updates, but risk of divergence

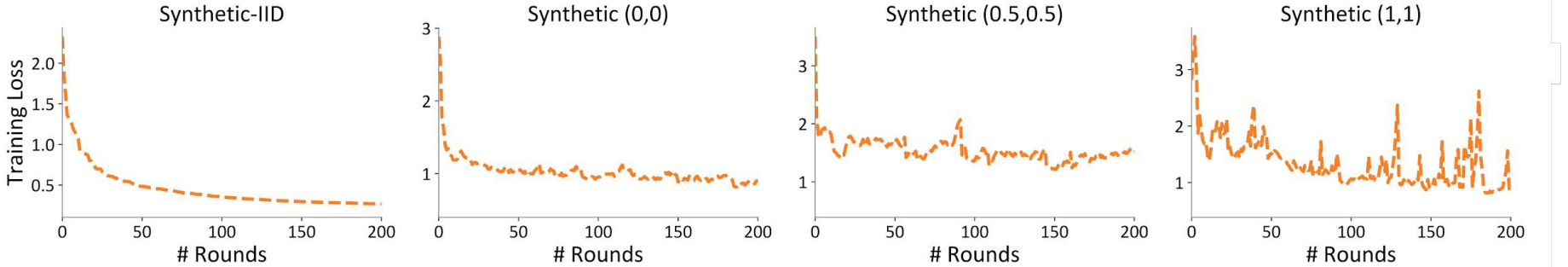
[Li et al, Federated optimization in heterogeneous networks, MLSys 2020]

WILL THIS CONVERGE?

Challenge: heterogeneity

•FedAvg is applied on synthetic datasets with increasing levels of heterogeneity:
- **Synthetic-IID**: All clients have data from the same distribution
- **Synthetic (0,0) → (1,1)**: Increasing non-IID-ness

effect of heterogeneity on FedAvg convergence, assuming all other hyperparameters fixed



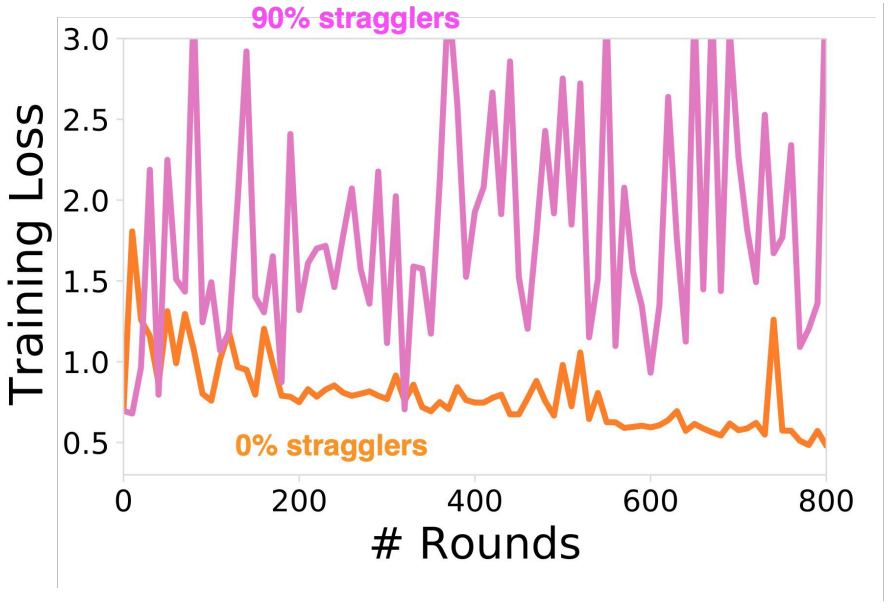
The more non-IID the data, the harder it is for FedAvg to converge — even if everything else is held constant.
This is because local model updates **diverge more from each other**, and their average no longer reflects true global progress.

[Li et al, Federated optimization in heterogeneous networks, MLSys 2020]

WILL THIS CONVERGE?

Challenge: heterogeneity

device stragglers (clients dropping or responding late) — negatively affects **FedAvg convergence**



systems heterogeneity (e.g., dropping devices*) can exacerbate convergence issues

*[Bonawitz, et al. Towards Federated Learning at Scale: System Design, MLSys, 2019]
[Li et al, Federated optimization in heterogeneous networks, MLSys 2020]

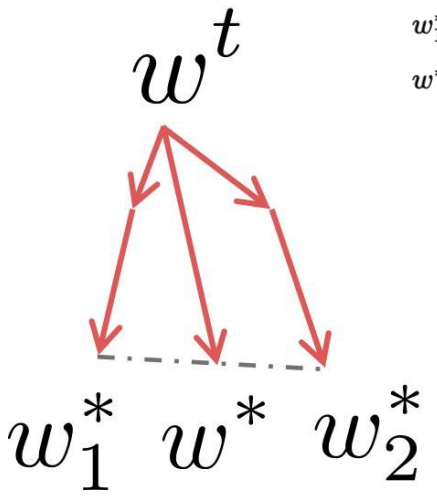
WILL THIS CONVERGE?

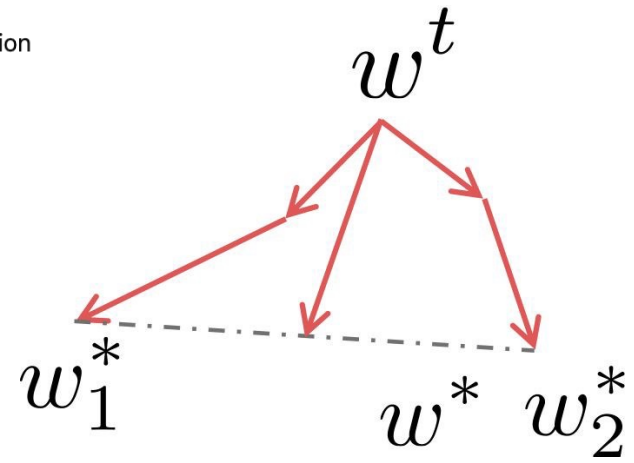
Federated Averaging (FedAvg)

w^t: current global model

w_1^*, w_2^*: local minima for clients 1 and 2

w^*: global (ideal) solution





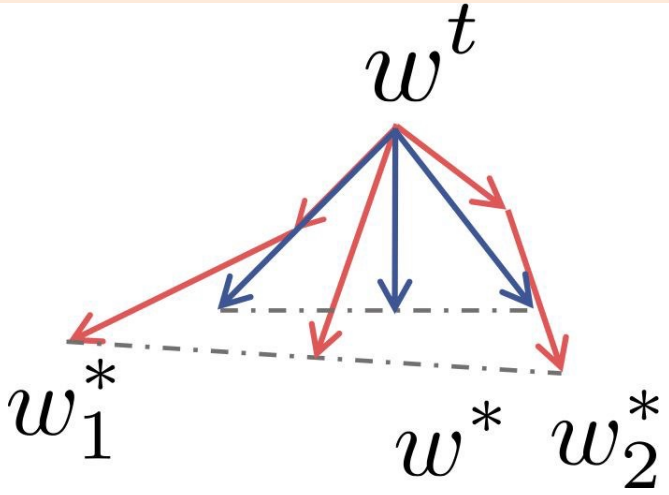
Local optima w1∗,w2∗ are **close to each other**
Updates from clients **pull in similar directions**

Local optima are **far apart** (due to **non-IID** data)
Updates **pull in conflicting directions**

Note that **FedAvg doesn't average the minima. It averages the gradients or updates** coming from training steps **that start at the current point w^t** — not the optima themselves.

Simple modification: FedProx

proximal term penalizing deviation from global model
•Each client update is **"anchored" to the global model**
•Clients still train locally, but are **discouraged from drifting too far** — especially important for **non-IID** data



$$\min_{w_k} F_k(w_k) + \frac{\mu}{2} \left\| w_k - w^t \right\|^2$$

proximal term

- proximal term *limits the impact of heterogeneous local updates*
- don't drop devices: instead [safely] incorporate partial work
- theoretical guarantees (with a dissimilarity assumption)

In vanilla FedAvg:
•If a **client doesn't finish** its local training (due to stragglers, disconnection, etc.):
- Its update is **dropped completely** — wasted effort.
- This leads to **biased updates** (only fast/available devices contribute).

In FedProx, we can still **use partial updates** from slow or interrupted devices.
The **proximal term** ensures their updates are **not too far from w^t**, so they **don't destabilize** the global model.

[Li et al, Federated optimization in heterogeneous networks, MLSys 2020]

Convergence guarantees

- $F_k(w)$ = Local loss function at client k.
- $f(w)$ = Global average loss function.
- $\nabla F_k(w)$ = Gradient at client k.
- $\nabla f(w)$ = Global gradient.

High-level: converges despite *non-IID data*, *local updating*, *partial participation*

- use *B-dissimilarity* notion to characterize statistical heterogeneity:

$$\mathbb{E}\left[\|\nabla F_k(w)\|^2\right] \leq \|\nabla f(w)\|^2 B^2$$

IID data: $B = 1$

non-IID data: $B > 1$

[Theorem] *Obtain suboptimality ε after T rounds, with*

- *Covers convex, non-convex losses*
- *Independent of local solver*

Upper bound on the number of iterations

$$T = O\left(\frac{f(w^0) - f^*}{\rho\varepsilon}\right)$$

a function of $(B, \mu, \ldots)$

[Yin, et al, Gradient Diversity: a Key Ingredient for Scalable Distributed Learning, AISTATS, 2018]

[Li et al, Federated optimization in heterogeneous networks, MLSys 2020]

Takeaways

Federated optimization methods that perform **local-updating** can significantly reduce communication rounds needed for convergence

However, heterogeneity can potentially lead to:

- slower convergence, reduced stability, divergence

Critical to *analyze* and *evaluate* federated methods with:

- non-IID data, partial / variable participation

A CLOSER LOOK

Federated optimization: challenges

Can **reduce communication** in federated optimization by:

1. Limiting *number of devices* involved in communication
2. Reducing number of *communication rounds*
3. **Reducing *size of messages* sent over network**

expensive communication

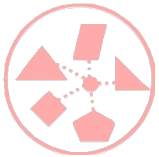
- massive, slow networks

privacy concerns

- user privacy constraints

statistical heterogeneity

- unbalanced, non-IID data



systems heterogeneity

- variable hardware, connectivity, etc

Communication-efficient FL

Can also consider techniques that reduce the size of messages (e.g., model updates) sent over the network

Common approaches:

- Dimensionality reduction (low-rank, sparsity)
  - Directly learn model updates that have reduced dimension/size
- Compression
  - Take regular (full dimension) updates and then compress them

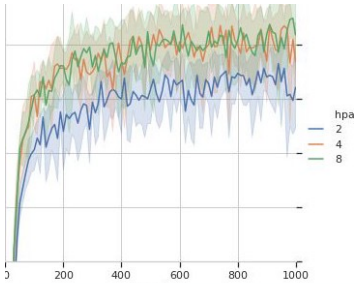
Can be applied to both uplink (model updates) and downlink (global model) messages

[Konečný et al, Federated Learning: Strategies for Improving Communication Efficiency, 2016]

Communication-efficient FL

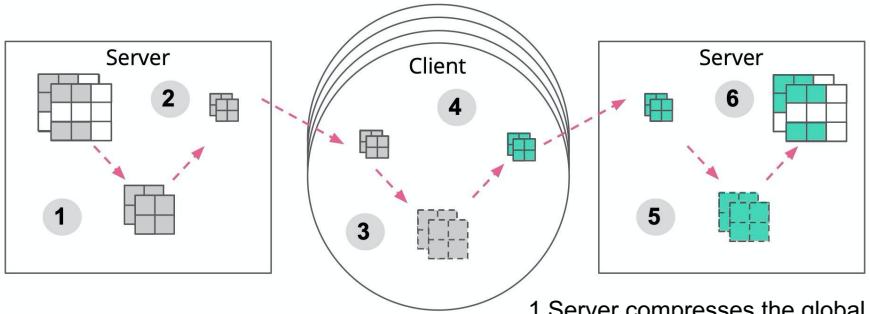
Compression techniques:

- Sparsification
- Quantization
- Dropout



Can empirically perform substantial quantization (e.g., 4 bits) without loss in accuracy (EMNIST dataset)

Can compose techniques:



1.Server compresses the global model.
2.Sends compressed model to the client.
3.Client trains locally.
4.Client compresses the update.
5.Sends compressed update back to server.
6.Server decompresses and aggregates updates to form a new global model.

Important to understand how compression may bias or increase variance in optimization procedure

[Caldas et al, Expanding the Reach of Federated Learning by Reducing Client Resource Requirements, 2018]

Hands-On Federated Optimization

TensorFlow Federated (TFF)

tensorflow.org/federated - github.com/tensorflow/federated

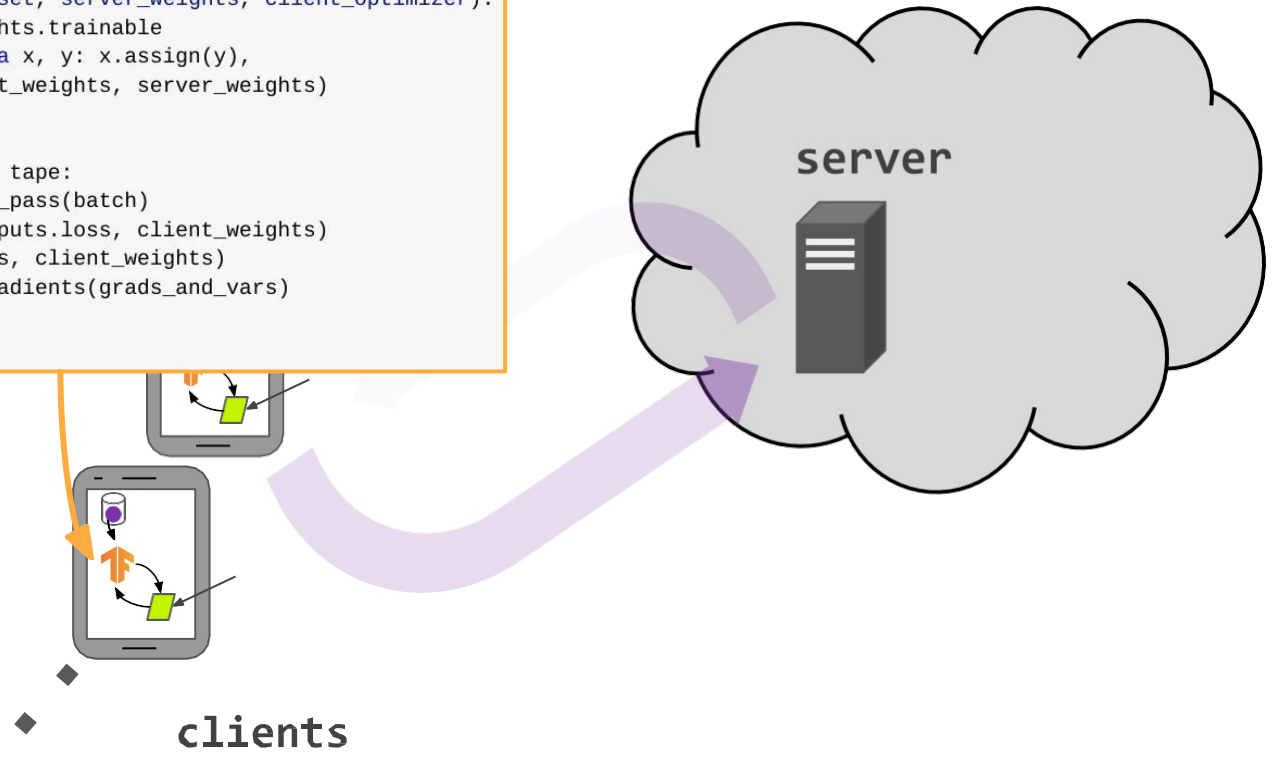
Declarative API and language for defining federated computations

High-performance distributed datacenter runtime providing scalable parallel execution of complex per-user computations

Computation libraries for federated learning and analytics, implementing sophisticated optimization algorithms and DP aggregations via TF Privacy.

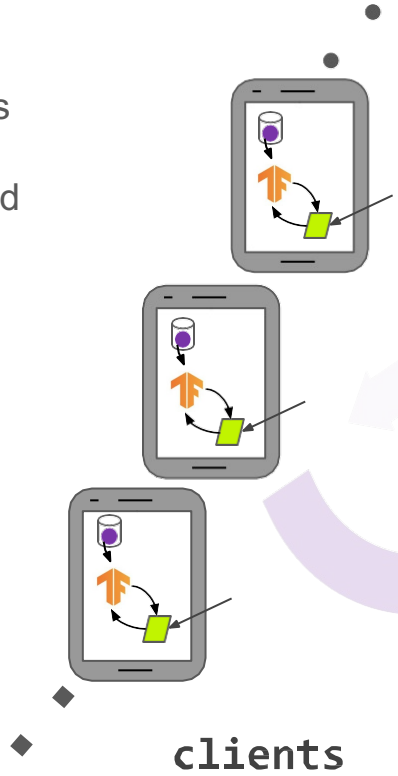
Federated Averaging in TFF: Client Computation

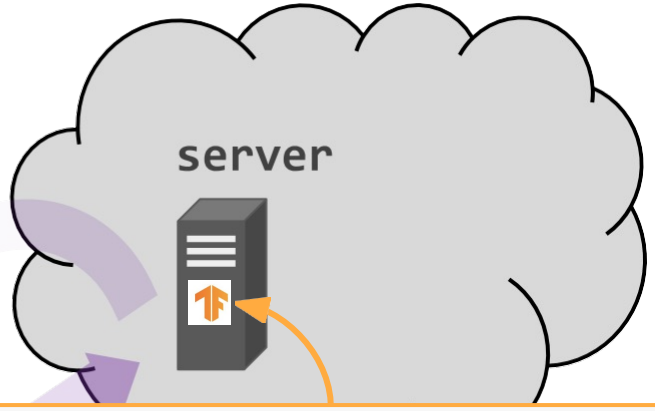
```
@tf.function
def client_update(model, dataset, server_weights, client_optimizer):
  client_weights = model.weights.trainable
  tf.nest.map_structure(lambda x, y: x.assign(y),
                        client_weights, server_weights)

  for batch in dataset:
    with tf.GradientTape() as tape:
      outputs = model.forward_pass(batch)
    grads = tape.gradient(outputs.loss, client_weights)
    grads_and_vars = zip(grads, client_weights)
    client_optimizer.apply_gradients(grads_and_vars)

  return client_weights
```



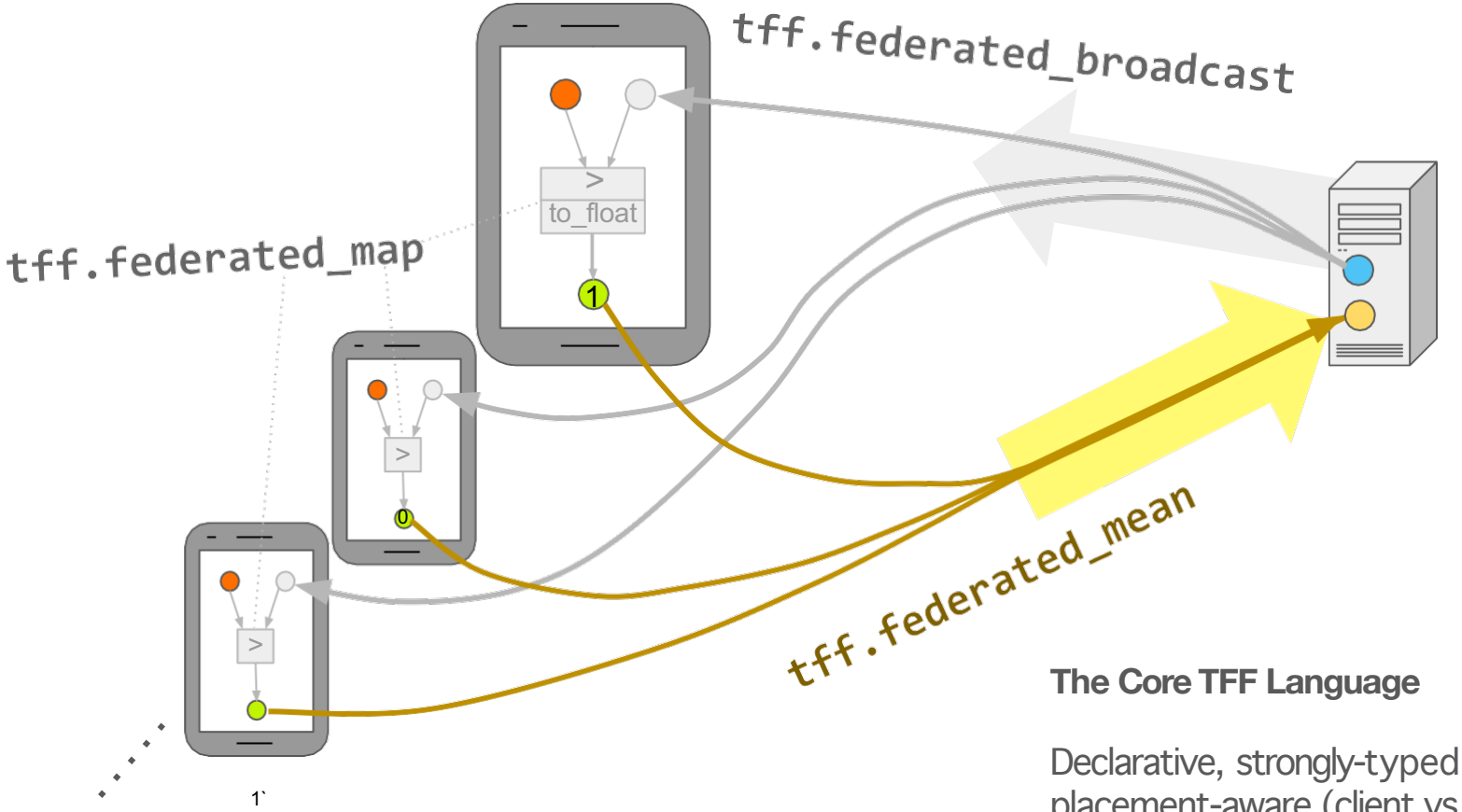
Federated Averaging in TFF: Server Computation

These TF computations can be serialized as computation graphs and shipped to real device.






```
@tf.function
def server_update(model, mean_client_update):
  model_weights = model.weights.trainable
  tf.nest.map_structure(lambda w, delta: w + server_learning_rate * delta,
                        model_weights, mean_client_update)
  return model_weights
```



The Core TFF Language

Declarative, strongly-typed,
placement-aware (client vs. server)

tff.federated_computation

Specification of a distributed computation, not code that runs in any one place.

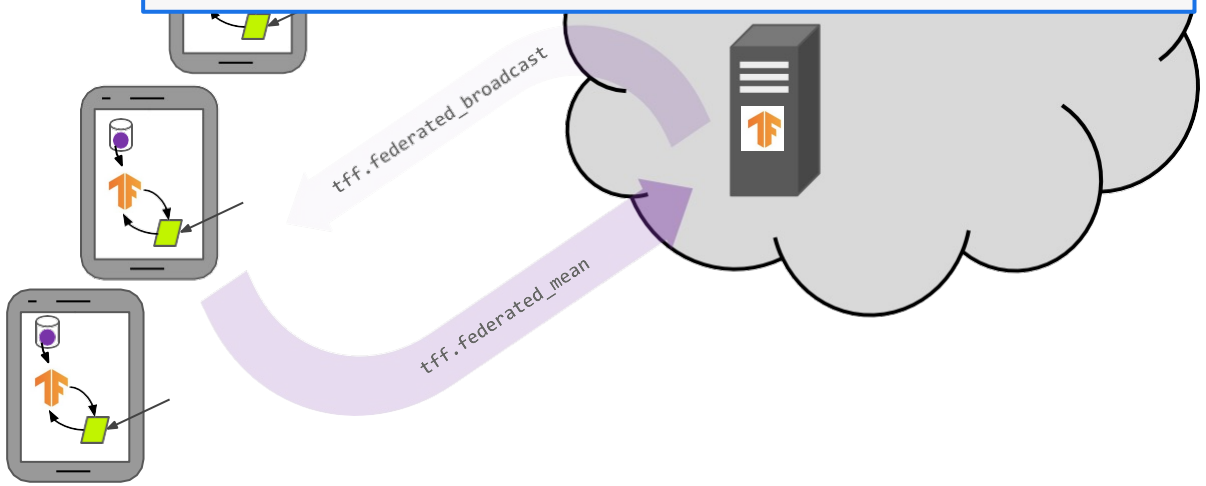
No enumeration or indexing of clients.

Values have types indicating @SERVER, @CLIENTS.

Think of the values as *handles* or *references* to data that may live in many places.

Declarative aggregation and broadcast operators amenable to large-scale multi-tier distributed architectures.

```
@tff.federated_computation(federated_server_type, federated_dataset_type)
def next_fn(server_weights, federated_dataset):
  server_weights_at_client = tff.federated_broadcast(server_weights)

  client_weights = tff.federated_map(
      client_update_fn, (federated_dataset, server_weights_at_client))

  mean_client_weights = tff.federated_mean(client_weights)

  server_weights = tff.federated_map(server_update_fn, mean_client_weights)

  return server_weights
```

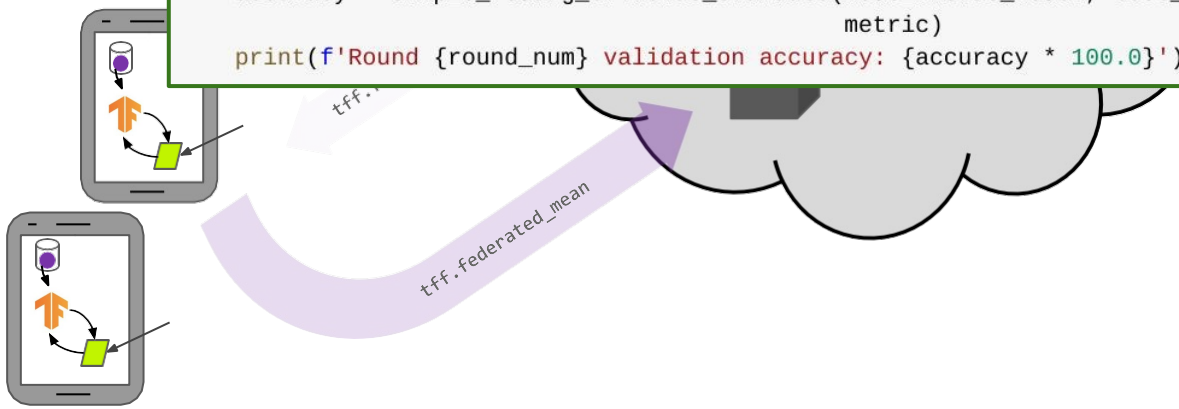


https://www.tensorflow.org/federated/tutorials/building_your_own_federated_learning_algorithm

TFF Simulation Runtime

Outer Python loop encodes the outer orchestration logic and simulates system factors (client availability, dropout, etc).

```
for round_num in range(total_rounds):
  sampled_clients = np.random.choice(
      train_data.client_ids,
      size=FLAGS.train_clients_per_round,
      replace=False)
  sampled_train_data = [
      train_data.create_tf_dataset_for_client(client)
      for client in sampled_clients
  ]
  server_state, train_metrics = iterative_process.next(
      server_state, sampled_train_data)
  print(f'Round {round_num} training loss: {train_metrics}')
  if round_num % FLAGS.rounds_per_eval == 0:
    model.from_weights(server_state.model_weights)
    accuracy = simple_fedavg_tf.keras_evaluate(model.keras_model, test_data,
                                               metric)
    print(f'Round {round_num} validation accuracy: {accuracy * 100.0}')
```



https://www.tensorflow.org/federated/tutorials/building_your_own_federated_learning_algorithm

Empirical Evaluation of Federated Optimization

Our focus

Federated learning offers the possibility to dramatically improve privacy versus utility trade-offs across a wide range of ML problems.

Federated learning can make the world a better place.

Fully realizing this potential will require new research and new algorithms.

Empirical evaluation of FL algorithms should help
***identify algorithms that can have the maximum impact in the real world**.*

Empirical studies can of course answer other questions as well, but the above goal is our focus in this section.

Challenges of evaluating FL algorithms

Most researchers (even at Google!) can't easily test new algorithms on a fleet of millions of mobile devices (cross-device) or a network of data centers around the world (cross-silo).

Evaluating federated learning algorithms requires a ***simulation***.

Ramifications:

- Need to distinguish between:
  - The real-world scenarios that motivate the work.
  - The details of the simulation performed.
- Both are important!

- *The results of the simulation need to plausibly predict performance on the real-world problem.*

Best practices for evaluating FL algorithms

- Be precise about the characteristics of applicable FL settings
  - Primarily: Cross-silo vs cross-device (stateful vs stateless algorithms)
- Be precise about what computations happen where, and what is communicated
  - Important for efficiency and privacy!
- Hyperparameter tuning matters
  - Tuning takes longer in the federated setting, so algorithms that "self-tune" are much prefered!
- Ensure reproducibility of simulations.
- Prefer real-world non-IID datasets
  - Pathological cases where labels are partitioned across clients aren't realistic
  - Is a single global model really what you want?
- Select metrics carefully

  "x-axis"
  - rounds of communication
  - bytes communicated
  - simulated time accounting for expected real-world compute & comm costs
  - (not the actual simulation wall-clock time)

  "y-axis"
  - accuracy or other domain-specific metrics on held-out test data
  - (not training set loss)

Best practices for evaluating FL algorithms

- Be precise about the characteristics of applicable FL settings
  - Primarily: Cross-silo vs cross-device (stateful vs stateless algorithms)

Here the focus is accuracy and efficiency. But other objectives matter as well:
Privacy, robustness, fairness, ability to personalize, …

We'll have more to say in Part IV.

- Ensure reproducibility of simulations.

- Prefer real-world non-IID datasets
  - Pathological cases where labels are partitioned across clients aren't realistic
  - Is a single global model really what you want?

- Select metrics carefully

"x-axis"

  - rounds of communication
  - bytes communicated
  - simulated time accounting for expected real-world compute & comm costs
  - (not the actual simulation wall-clock time)

"y-axis"

  - accuracy or other domain-specific metrics on held-out test data
  - (not training set loss)

Adaptive Federated Optimization

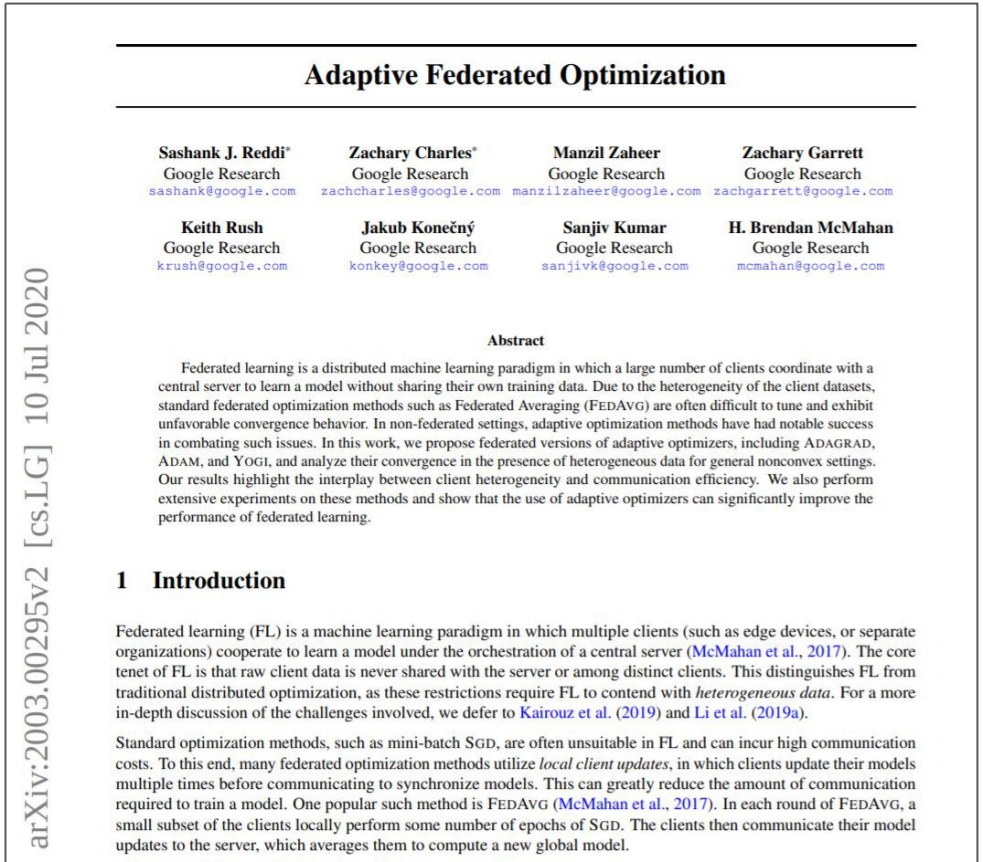
Extends FedAvg to a **new family of optimization algorithms**

Theory for server-side adaptive update rules (AdaGrad, Adam, Yogi)

Extensible implementation using TensorFlow Federated

Extensive empirical evaluation

Adaptive methods are optimization algorithms that adjust the learning rate **dynamically** for each parameter during training. In Federated Learning (FL), they are applied **on the server** during model aggregation.



Adaptive Federated Optimization

Sashank J. Reddi*
Google Research
sashank@google.com

Zachary Charles*
Google Research
zachcharles@google.com

Manzil Zaheer
Google Research
manzilzaheer@google.com

Zachary Garrett
Google Research
zachgarrett@google.com

Keith Rush
Google Research
krush@google.com

Jakub Konečný
Google Research
konkey@google.com

Sanjiv Kumar
Google Research
sanjivk@google.com

H. Brendan McMahan
Google Research
mcmahan@google.com

Abstract

Federated learning is a distributed machine learning paradigm in which a large number of clients coordinate with a central server to learn a model without sharing their own training data. Due to the heterogeneity of the client datasets, standard federated optimization methods such as Federated Averaging (FEDAVG) are often difficult to tune and exhibit unfavorable convergence behavior. In non-federated settings, adaptive optimization methods have had notable success in combating such issues. In this work, we propose federated versions of adaptive optimizers, including ADAGRAD, ADAM, and YOGI, and analyze their convergence in the presence of heterogeneous data for general nonconvex settings. Our results highlight the interplay between client heterogeneity and communication efficiency. We also perform extensive experiments on these methods and show that the use of adaptive optimizers can significantly improve the performance of federated learning.

1 Introduction

Federated learning (FL) is a machine learning paradigm in which multiple clients (such as edge devices, or separate organizations) cooperate to learn a model under the orchestration of a central server (McMahan et al., 2017). The core tenet of FL is that raw client data is never shared with the server or among distinct clients. This distinguishes FL from traditional distributed optimization, as these restrictions require FL to contend with *heterogeneous data*. For a more in-depth discussion of the challenges involved, we defer to Kairouz et al. (2019) and Li et al. (2019a).

Standard optimization methods, such as mini-batch SGD, are often unsuitable in FL and can incur high communication costs. To this end, many federated optimization methods utilize *local client updates*, in which clients update their models multiple times before communicating to synchronize models. This can greatly reduce the amount of communication required to train a model. One popular such method is FEDAVG (McMahan et al., 2017). In each round of FEDAVG, a small subset of the clients locally perform some number of epochs of SGD. The clients then communicate their model updates to the server, which averages them to compute a new global model.

why adaptive methods help?

01

In **federated learning**, data is **non-IID** (different users have different patterns).

02

FedAvg just averages all client updates.
➔ But if client updates are *biased*, averaging blindly can **slow convergence**.

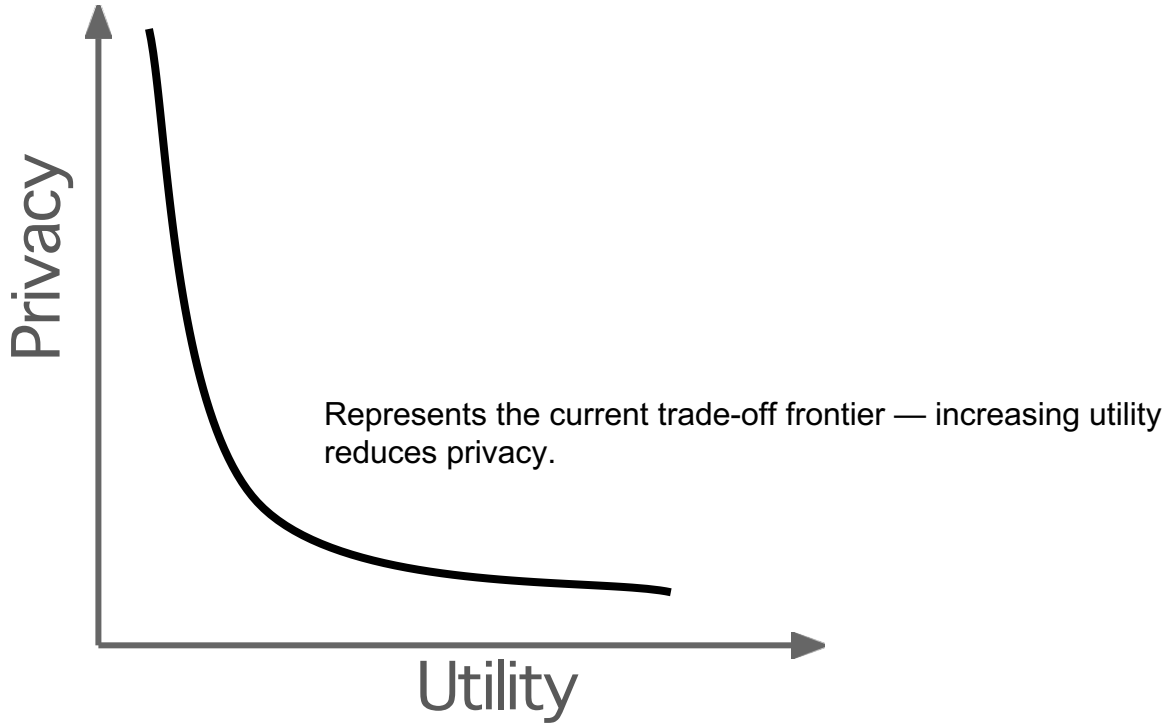
03

Adaptive methods (FedAdam, FedYogi, etc.) **adjust the server updates** based on past gradients (like Adam does in standard deep learning). ➔ Helps the server **stabilize** and **speed up** convergence despite non-IID updates!

Part III: Privacy for Federated Learning and Analytics

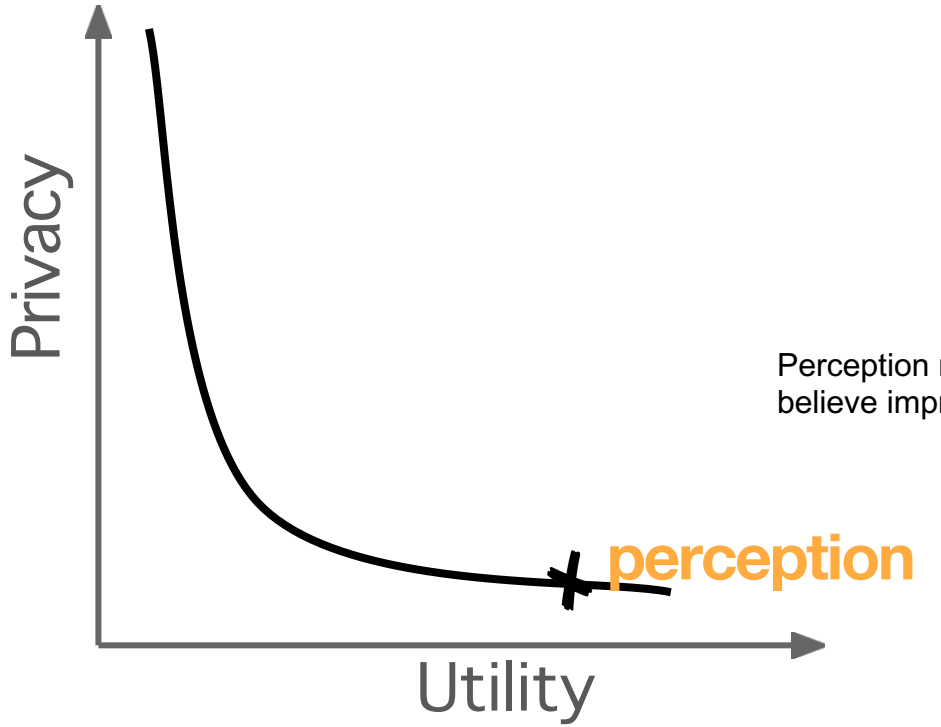
ML on sensitive data: privacy vs. utility

- **X-axis:** Utility (e.g., model accuracy, performance)
- **Y-axis:** Privacy (e.g., protection of user data)



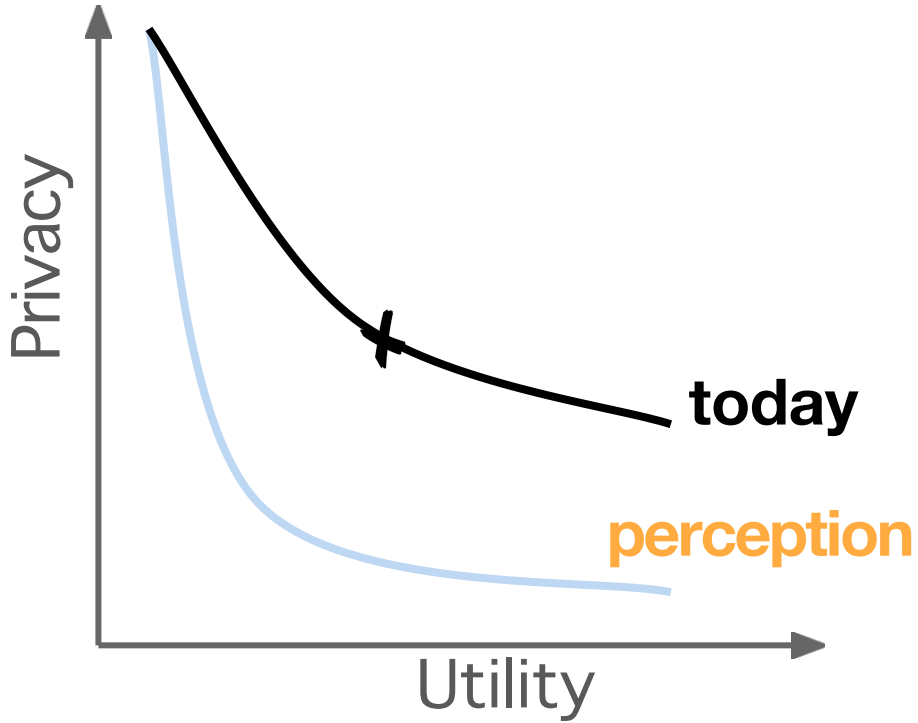
Represents the current trade-off frontier — increasing utility reduces privacy.

ML on sensitive data: privacy vs. utility



Perception represents how people perceive the trade-off — they believe improving utility comes with a much sharper privacy cost.

ML on sensitive data: privacy vs. utility

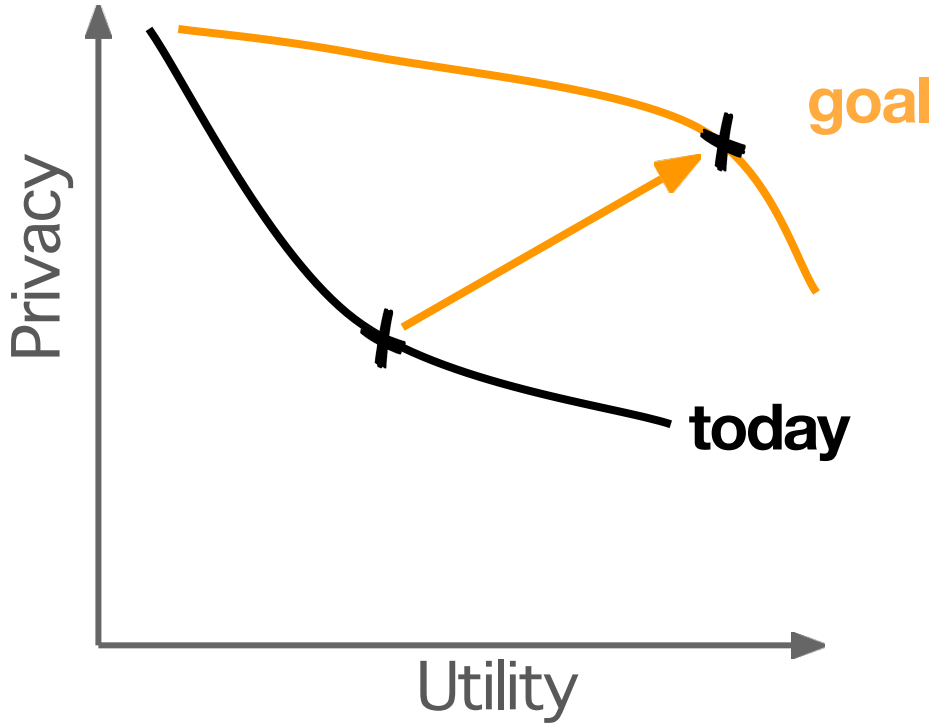


1. Policy
2. Technology

We can achieve better privacy–utility trade-offs than people think, with the right tools and regulations (e.g., federated learning, differential privacy).

ML on sensitive data: privacy vs. utility (?)

We don't have to choose between privacy and utility — with better technology (like federated learning, differential privacy), we can shift the entire trade-off curve.



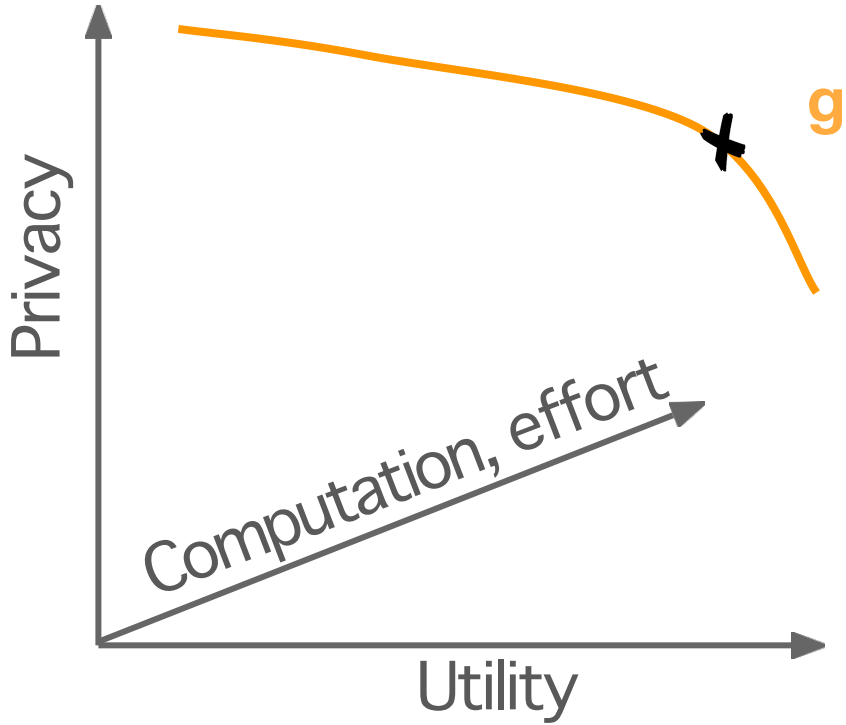
1. Policy
2. **New Technology**

Push the pareto frontier with better technology.

Make achieving high privacy and utility possible.

ML on sensitive data: privacy vs. utility (?)

A **Pareto frontier** shows the set of **optimal trade-offs** between two (or more) competing objectives.

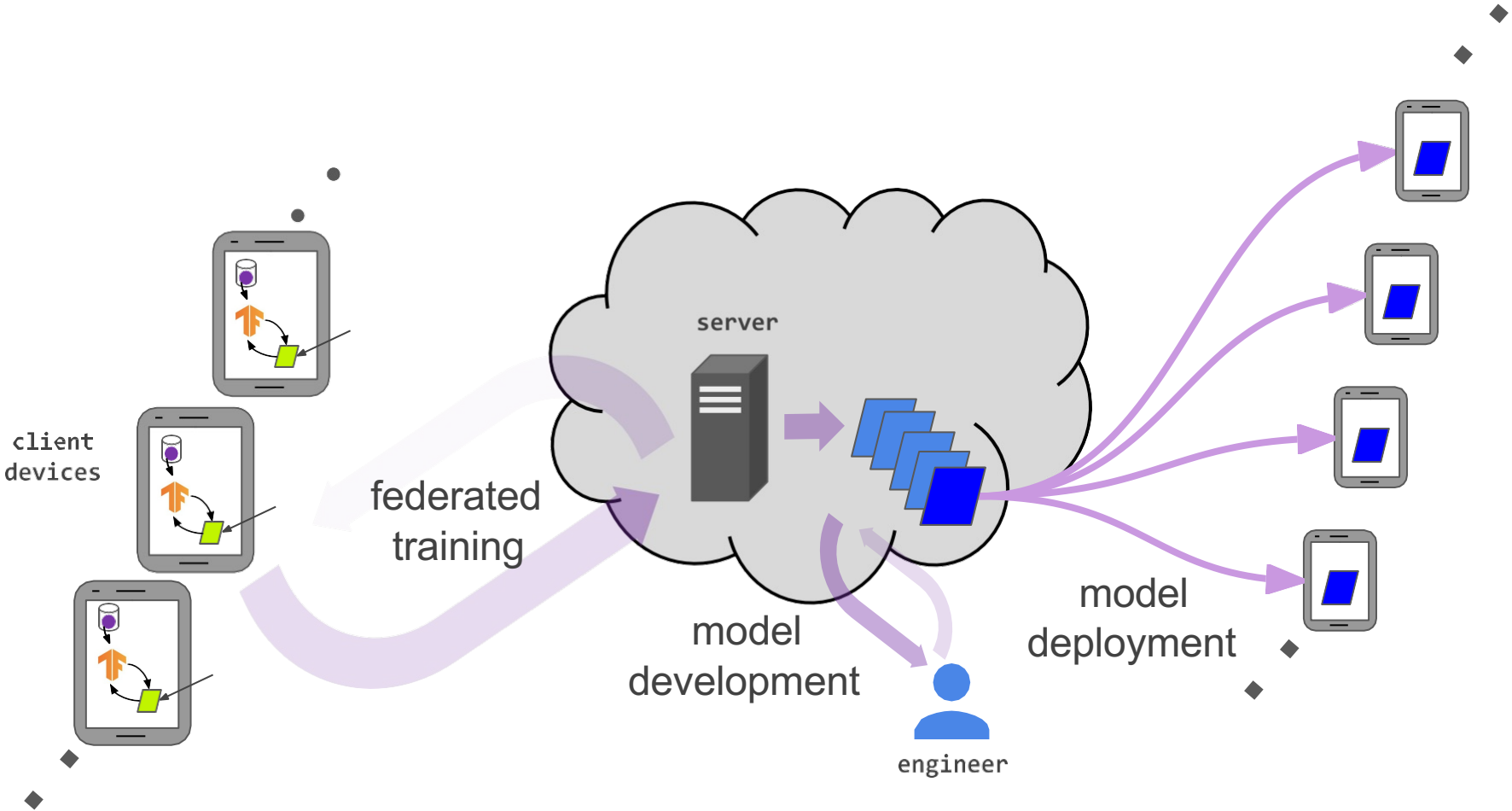


goal

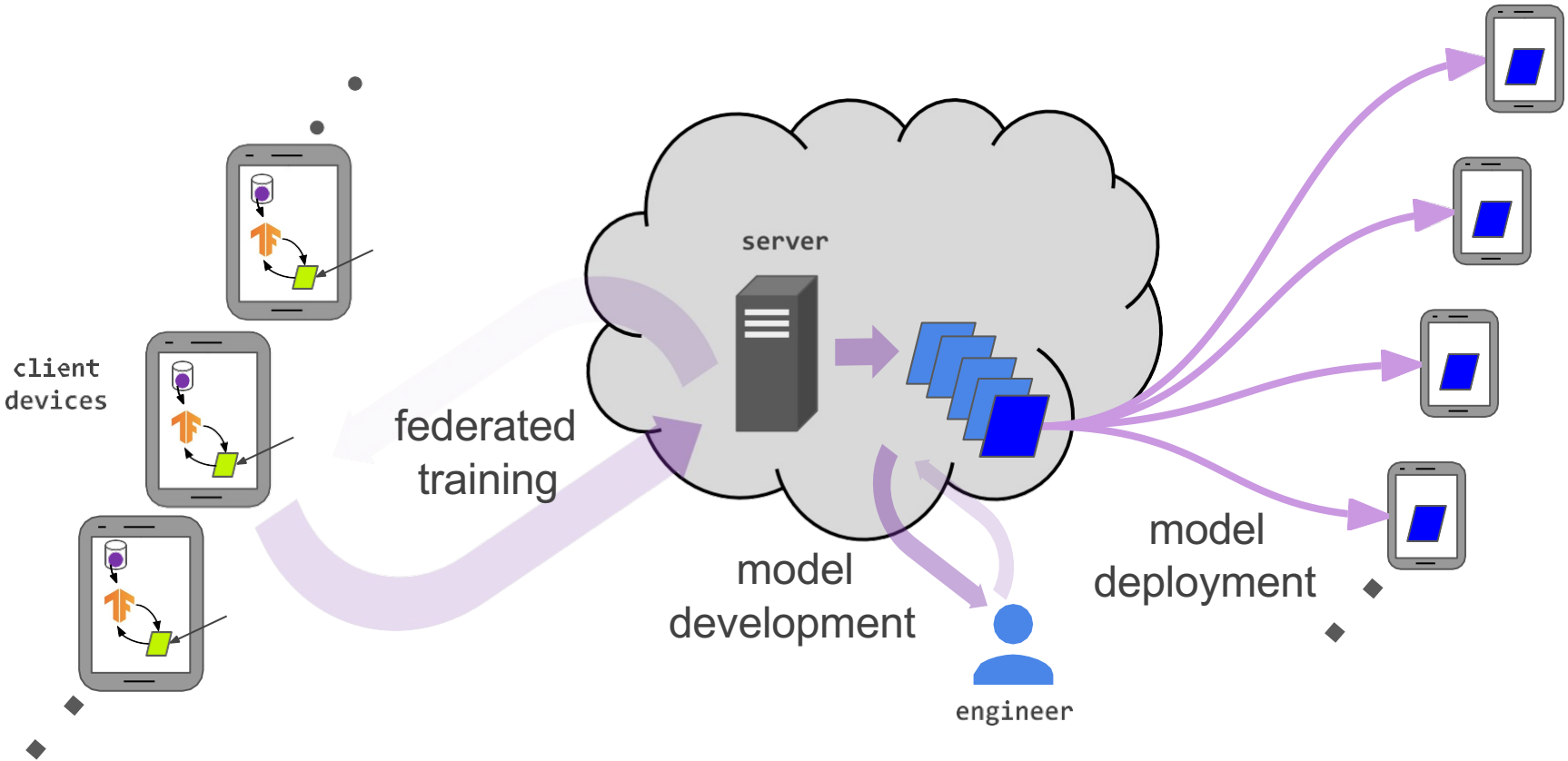
1. Policy
2. **New Technology**

Push the pareto frontier with better technology.

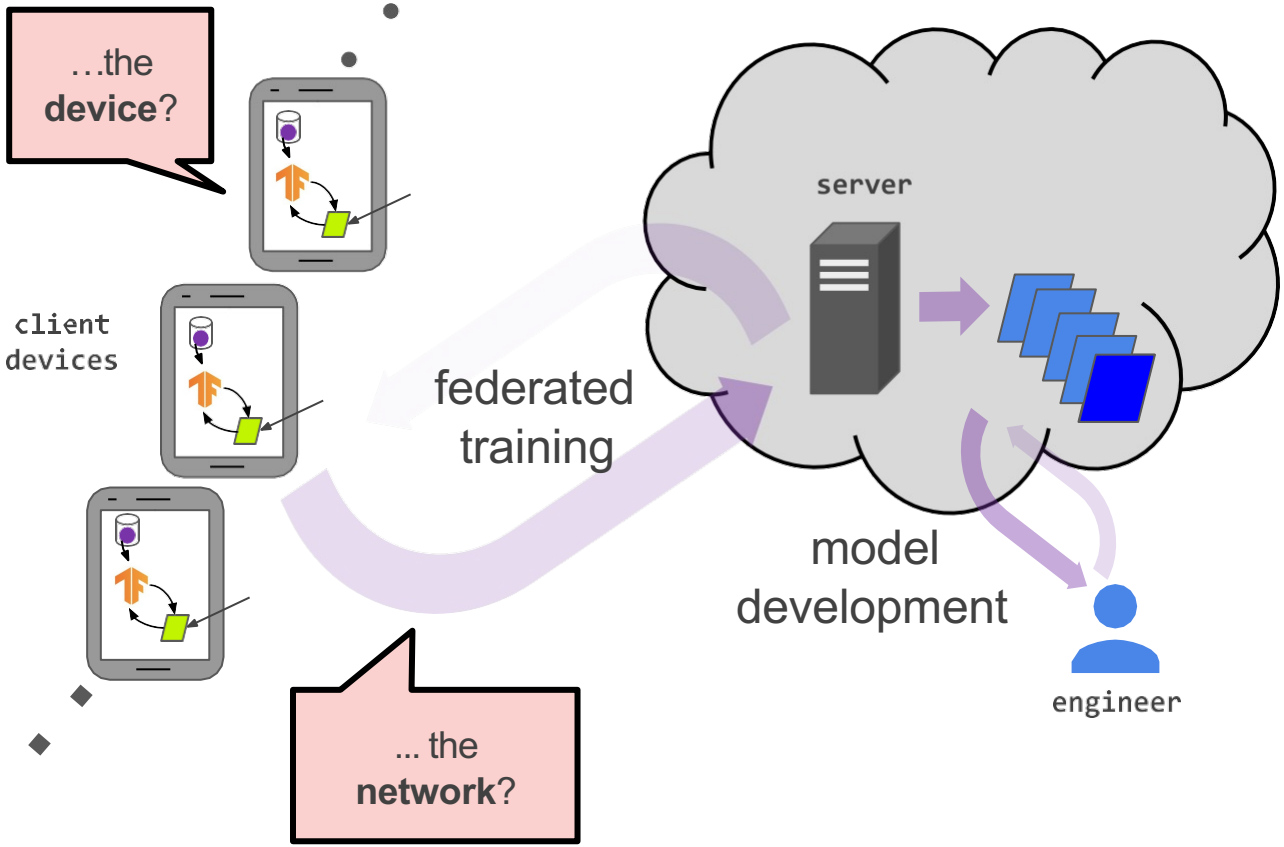
Make achieving high privacy and utility possible **with less work.**


client
devices
federated
training
server
model
development
engineer
model
deployment

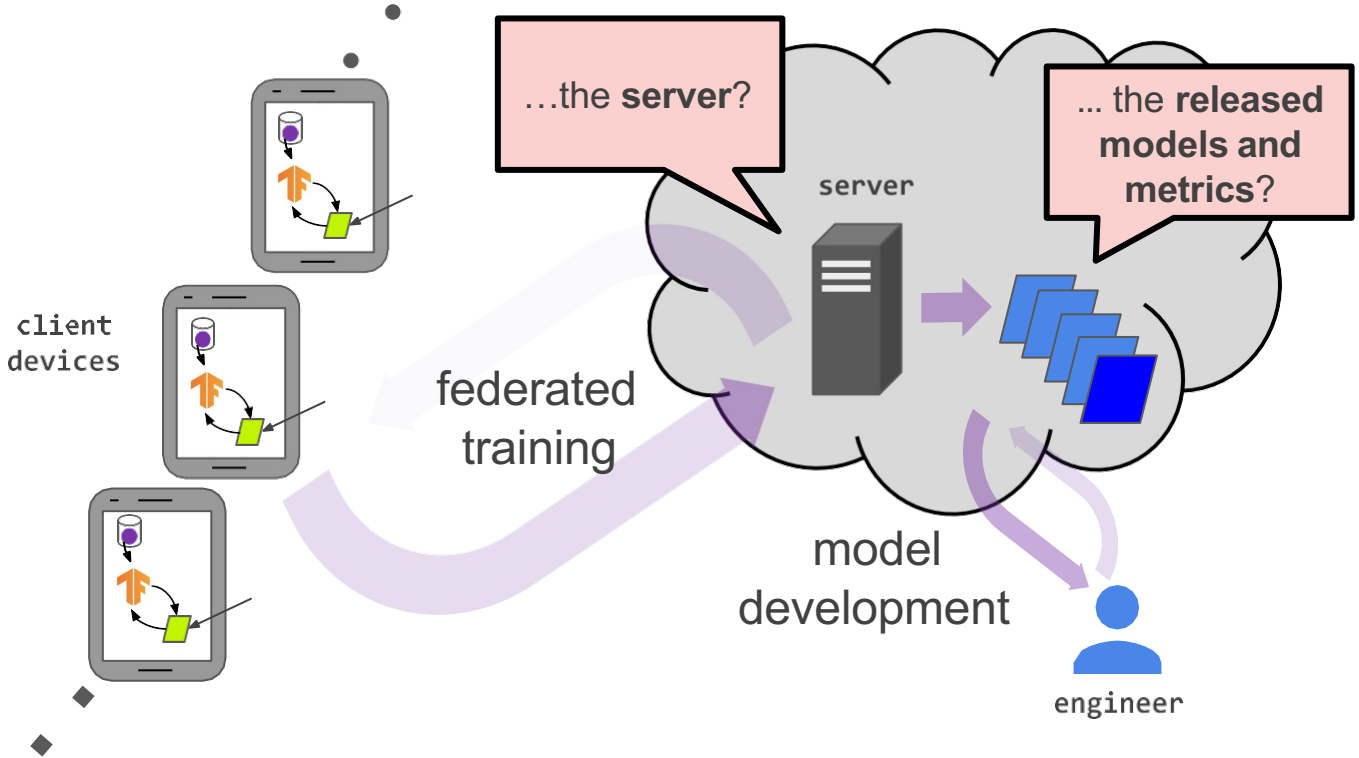
What private information might an actor learn?



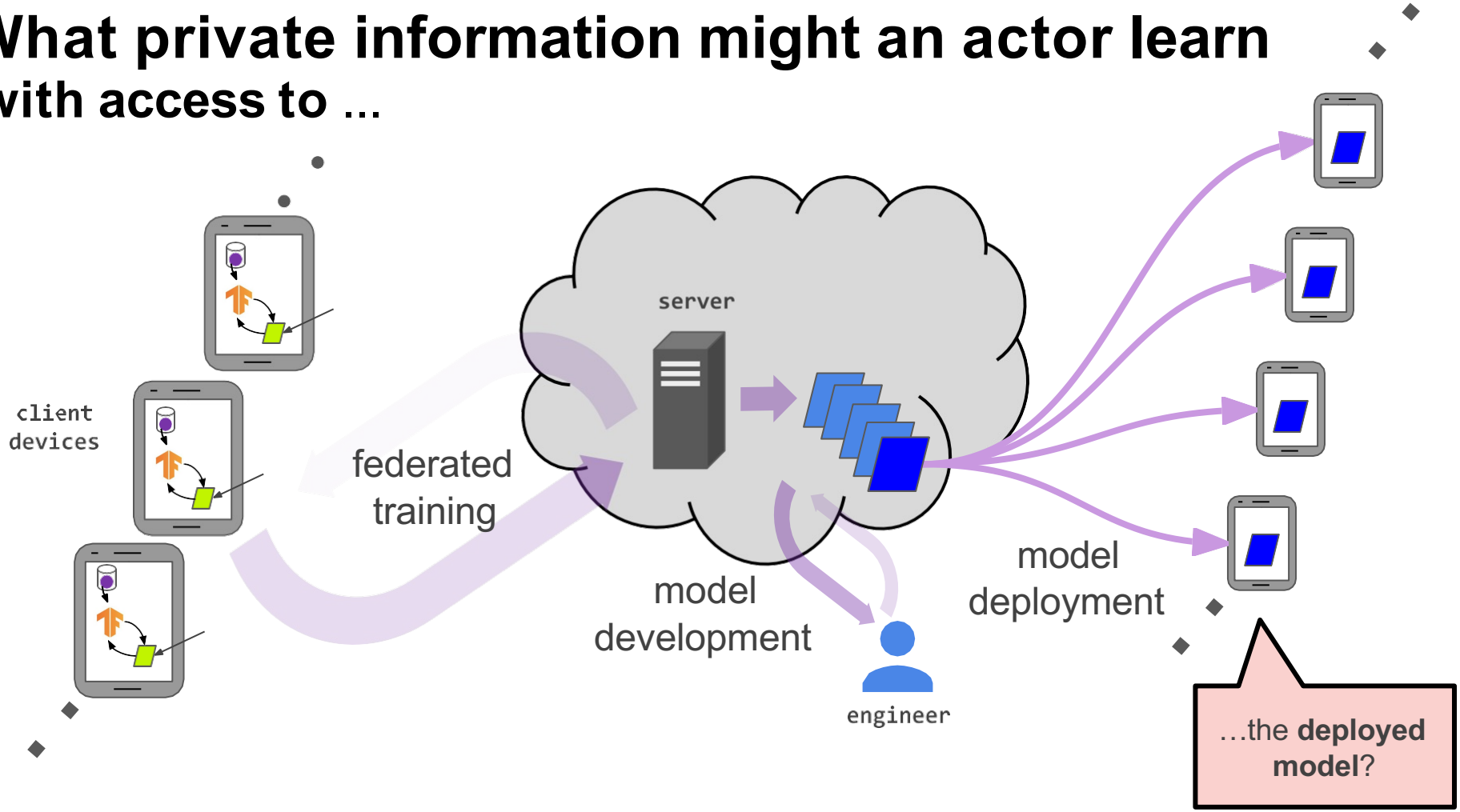
What private information might an actor learn with access to ...



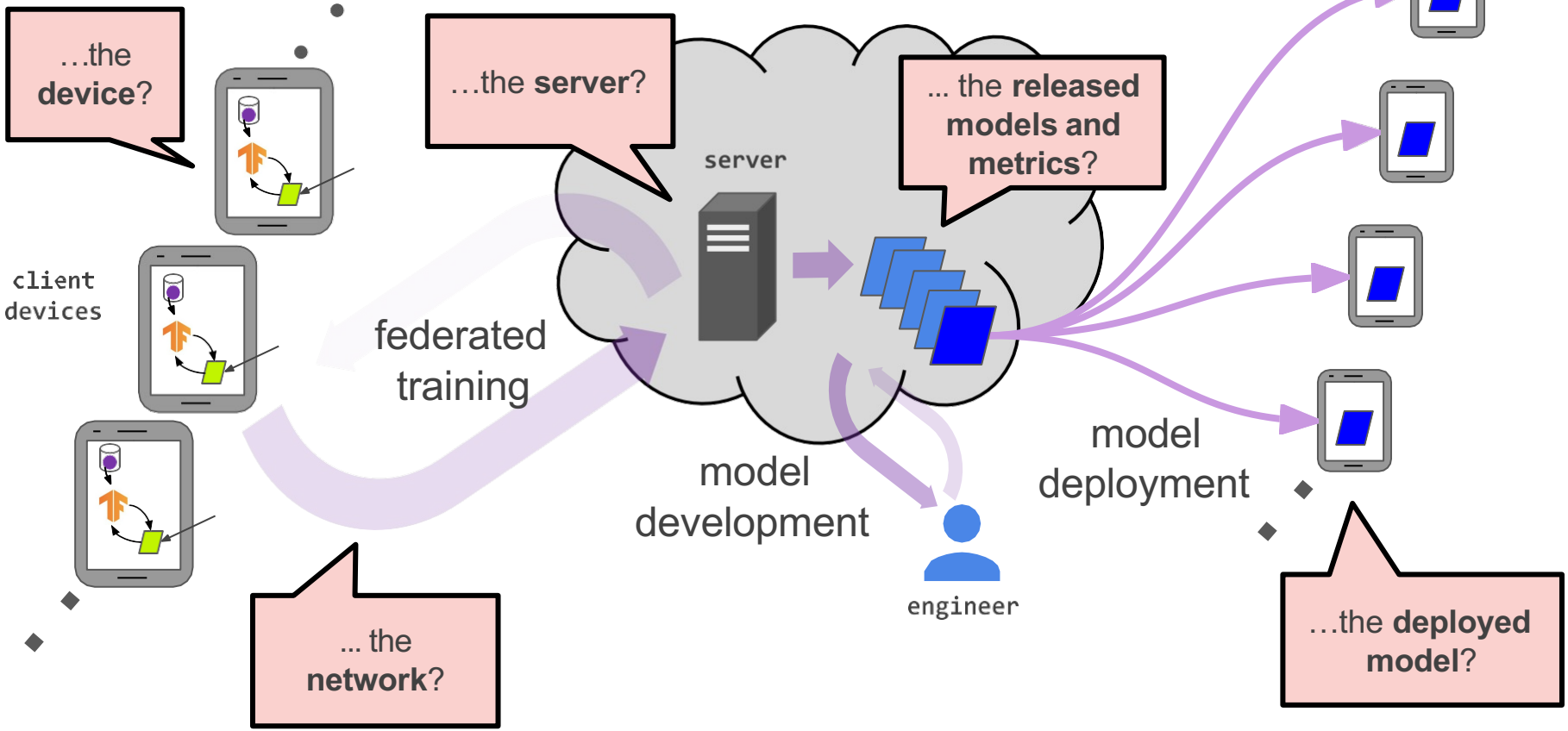
What private information might an actor learn with access to ...



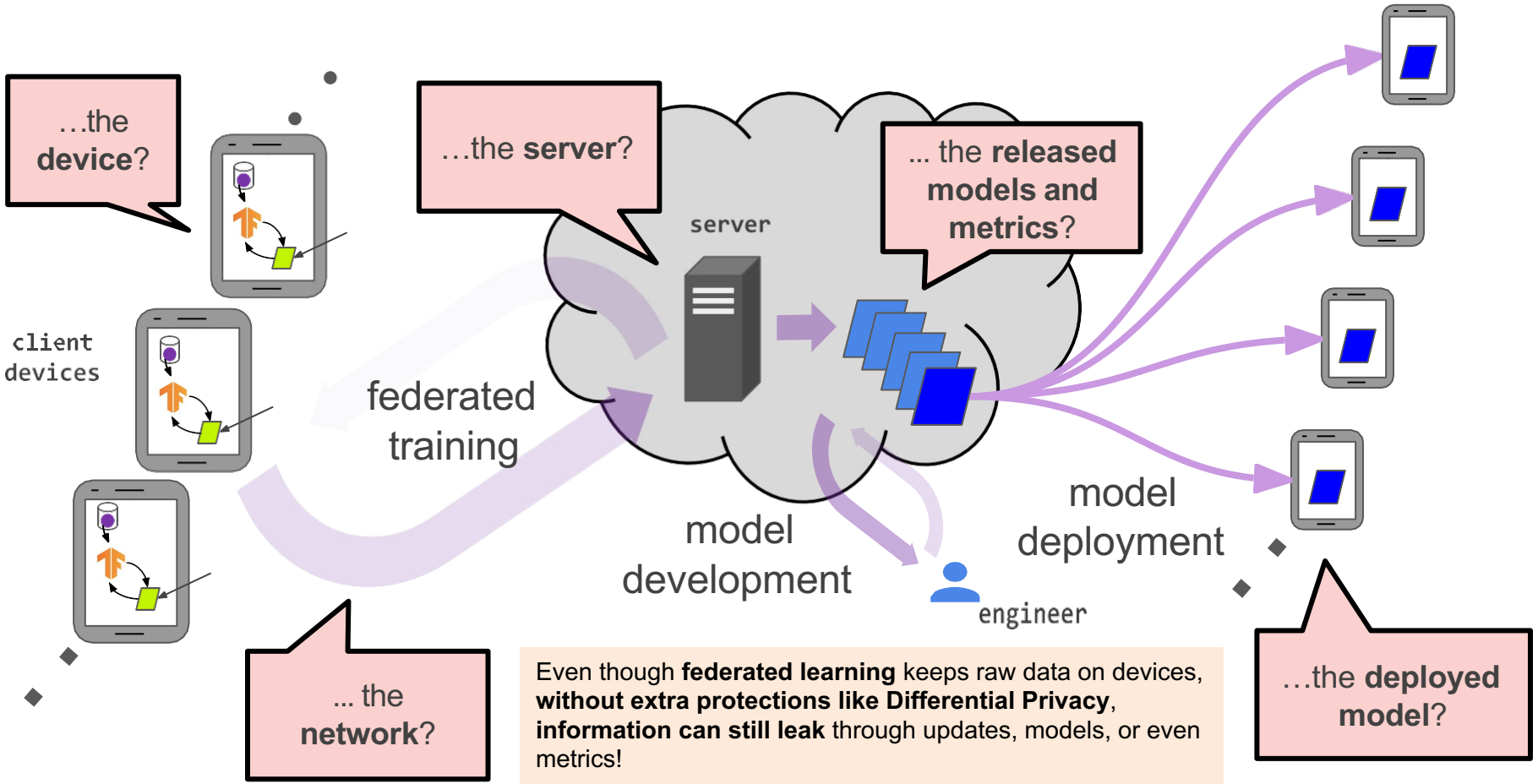
What private information might an actor learn with access to ...



What private information might an actor learn with access to ...

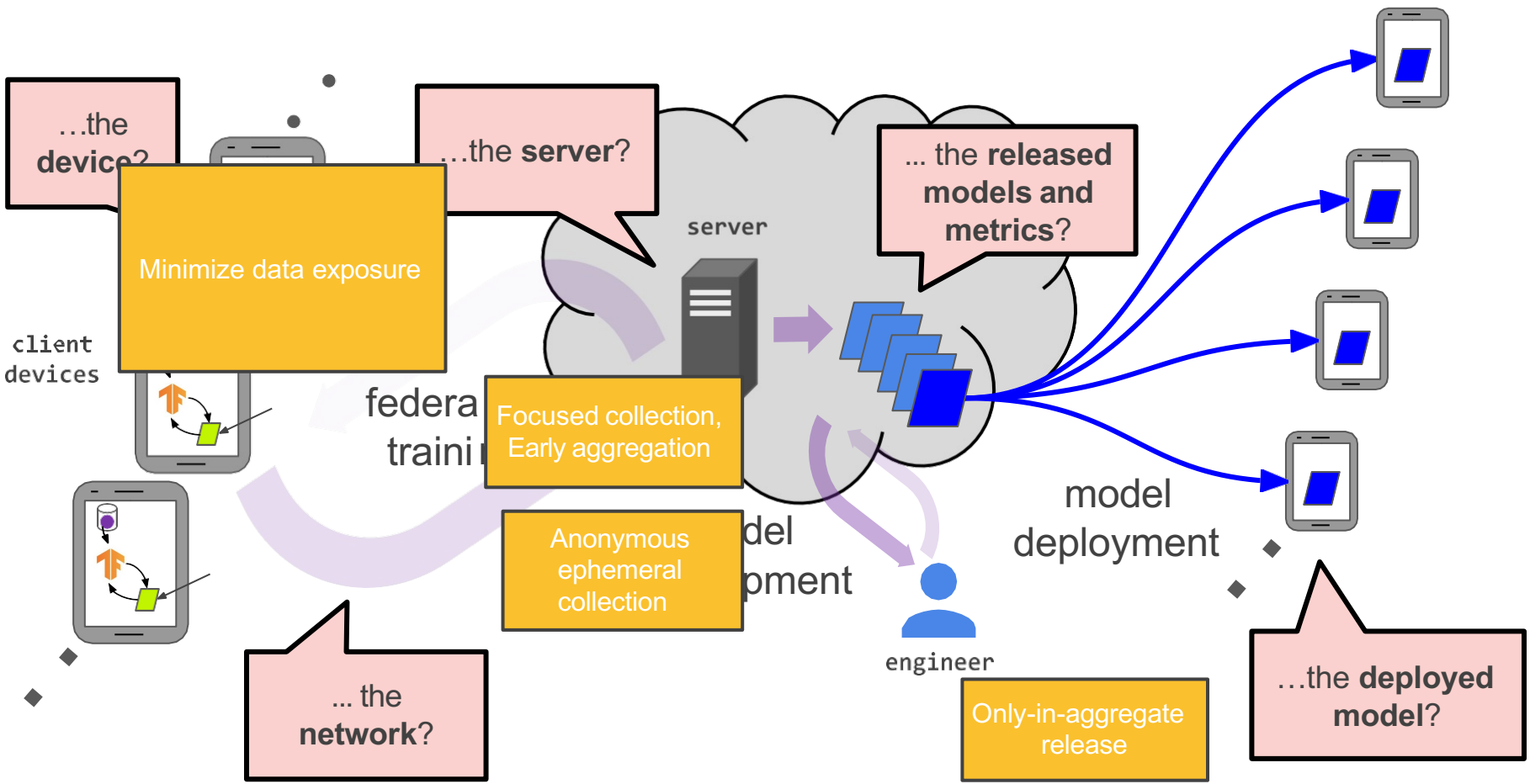


How much do I need to trust …

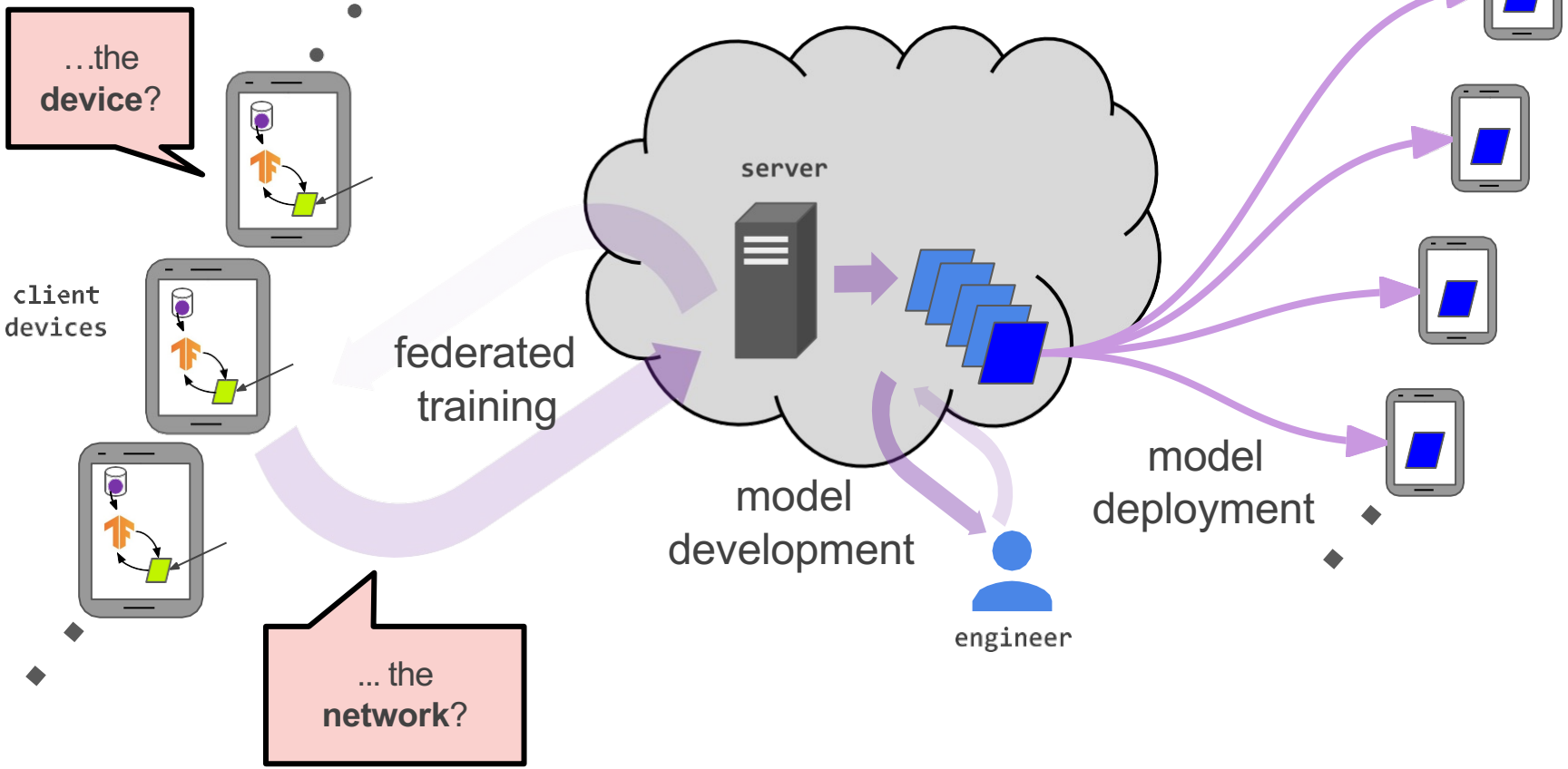


Even though **federated learning** keeps raw data on devices, **without extra protections like Differential Privacy**, **information can still leak** through updates, models, or even metrics!

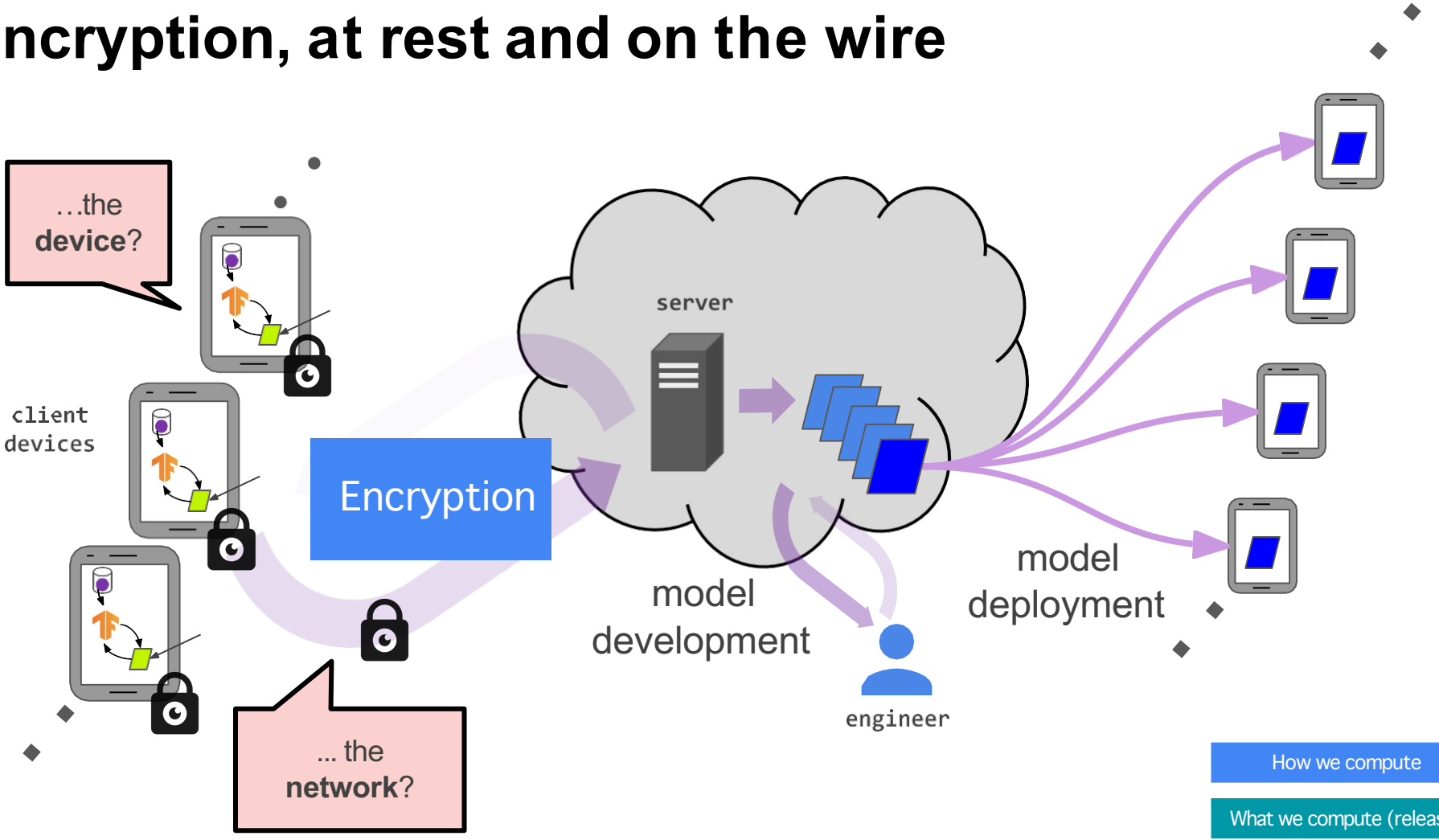
Privacy principles guiding FL



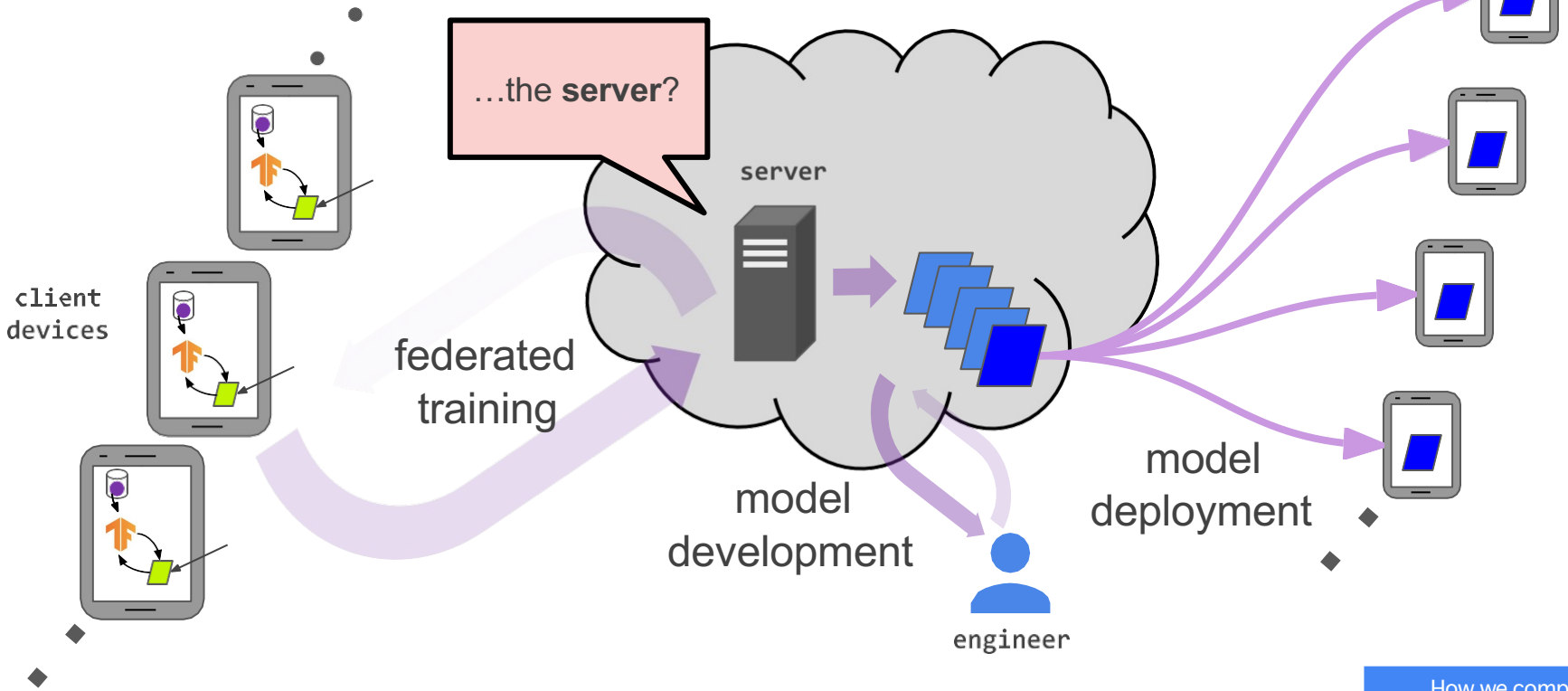
What private information might an actor learn with access to ...



Encryption, at rest and on the wire



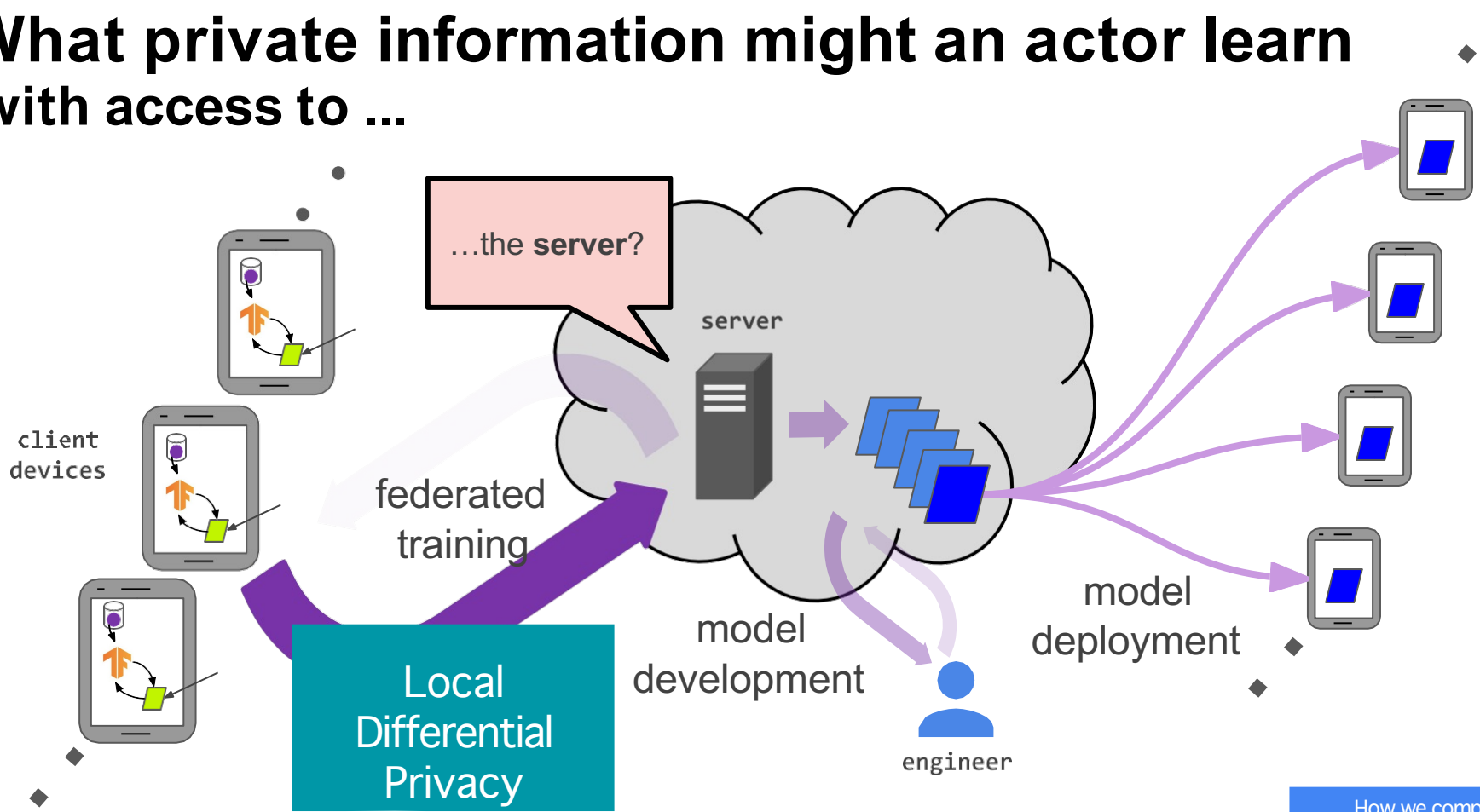
What private information might an actor learn with access to ...



How we compute

What we compute (release)

What private information might an actor learn with access to ...



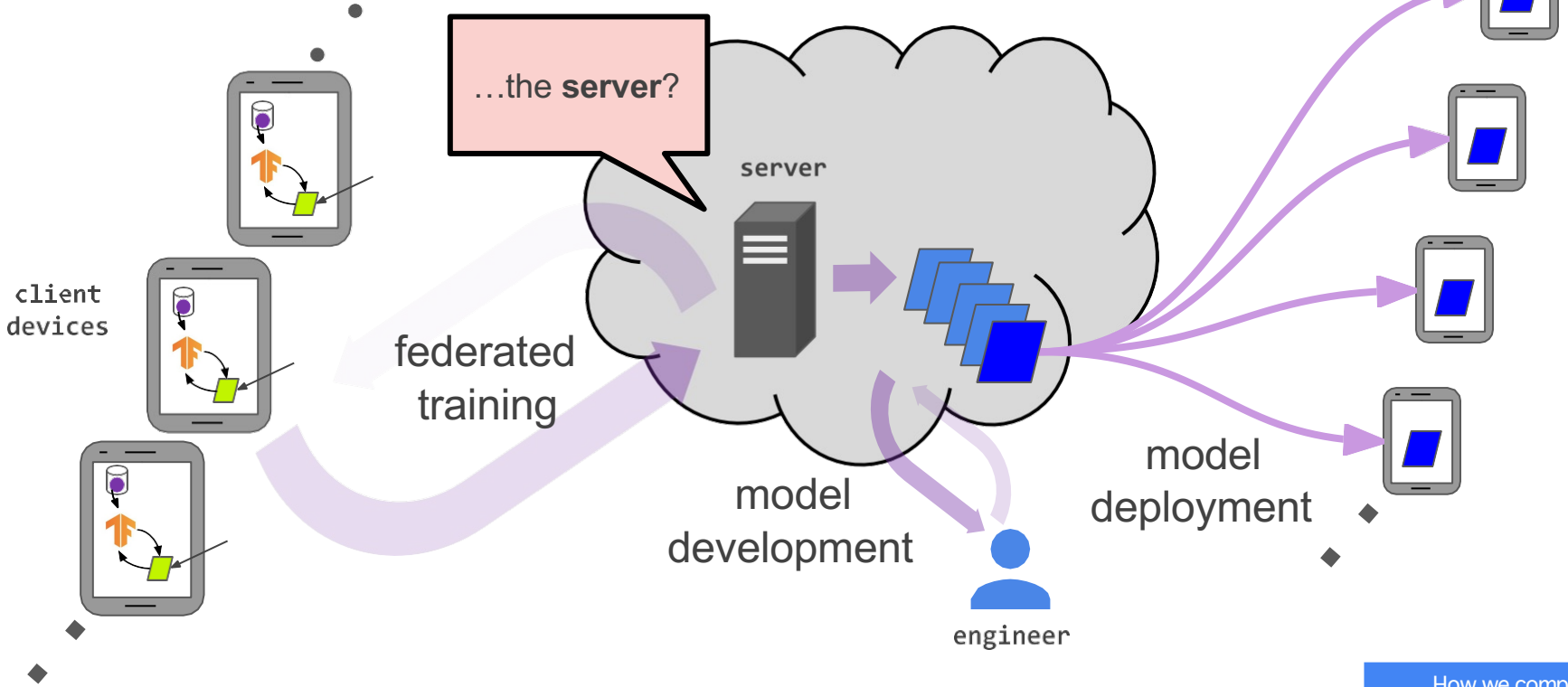
Each **client device adds noise** to its update **before** sending it to the server. The server only sees **noisy**, **private-safe updates** — not the exact data.

Warner. **Randomized response.** 1965.
Kasiviswanathan, et. al. **What can we learn privately?** 2011.

How we compute

What we compute (release)

What private information might an actor learn with access to ...

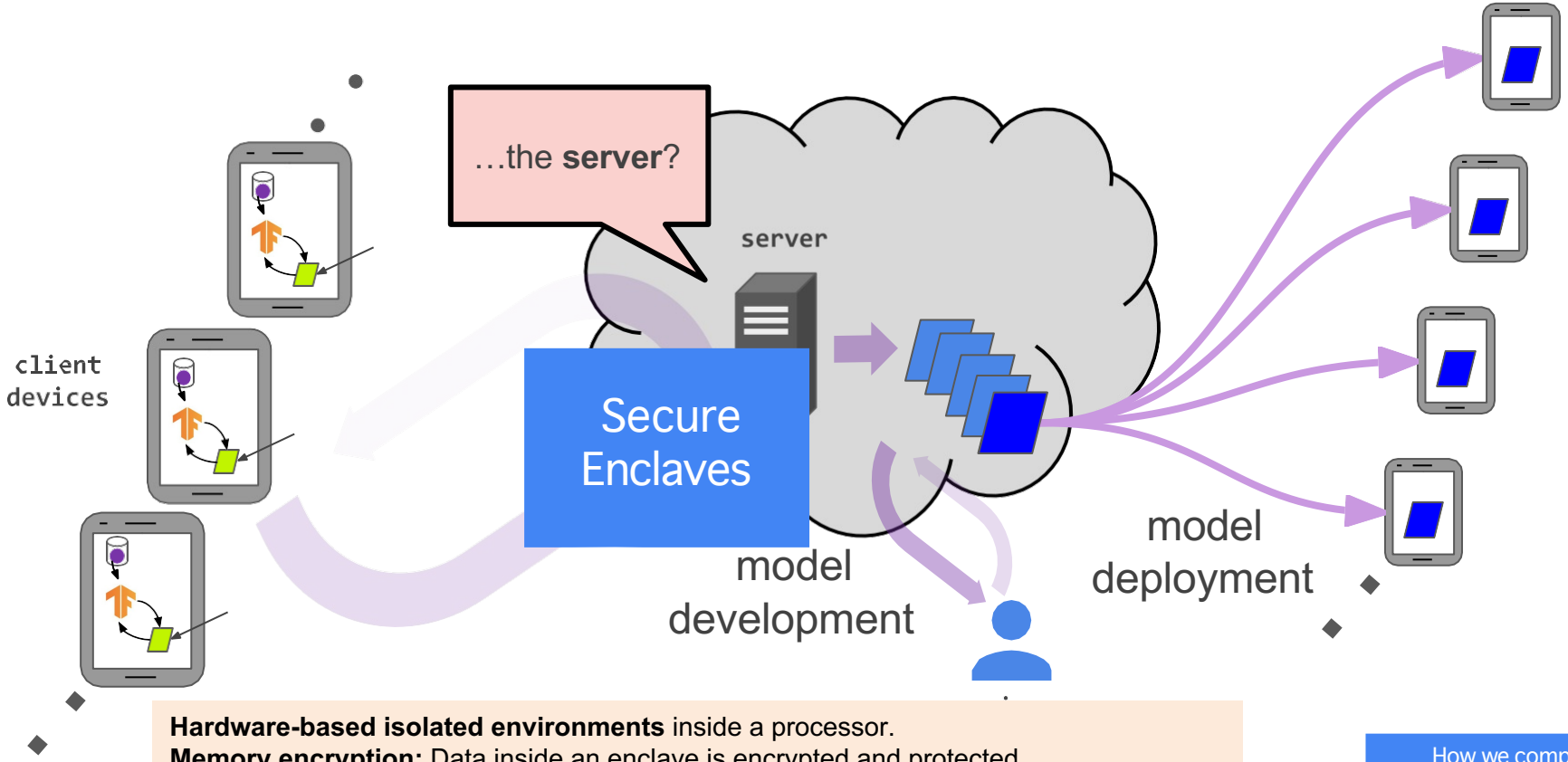


Ideally, **nothing**, even with root access.

How we compute

What we compute (release)

Trusted execution environments

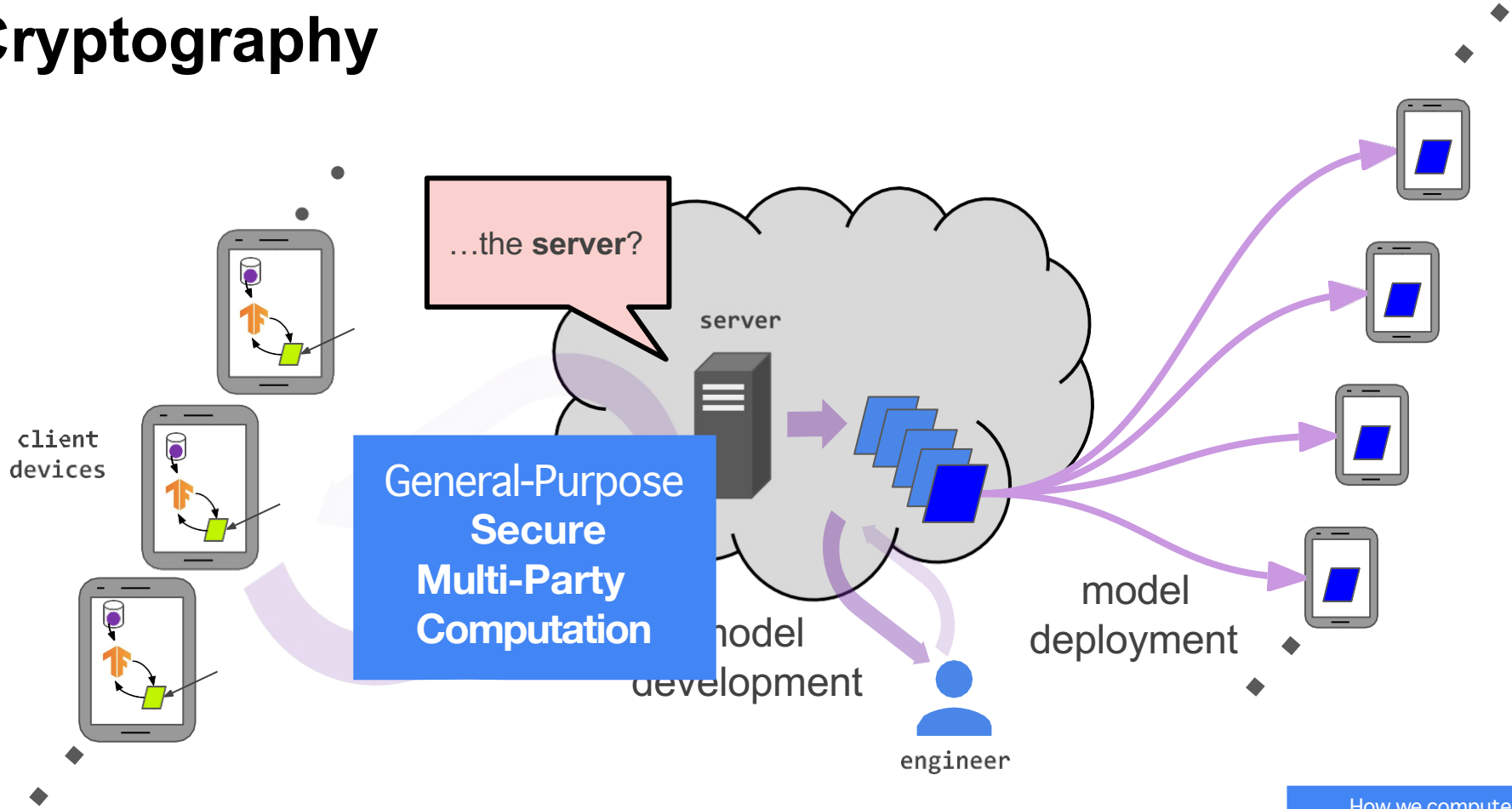


Hardware-based isolated environments inside a processor.
Memory encryption: Data inside an enclave is encrypted and protected.
Access control: Only specific trusted code can access the enclave data.
You don't have to trust the whole server — you only trust the small secure enclave.

How we compute

What we compute (release)

Cryptography

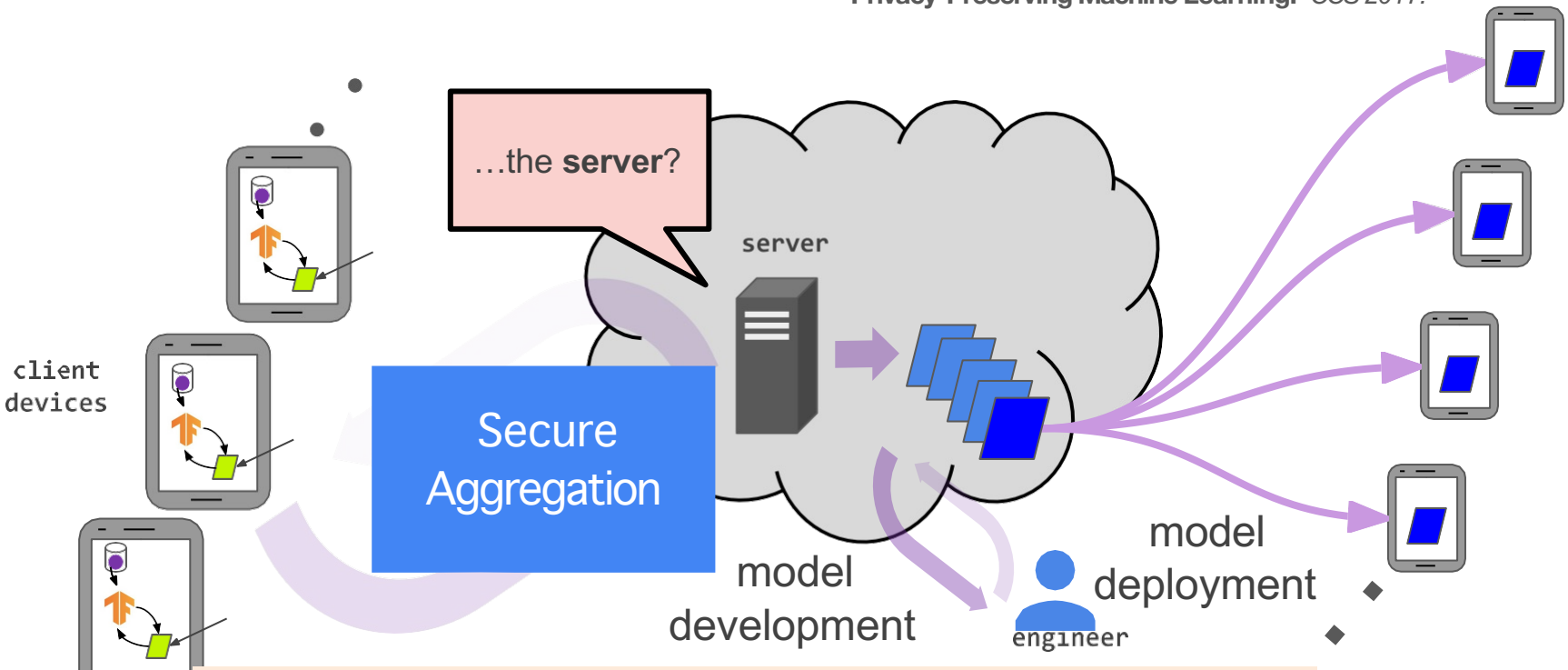


How we compute

What we compute (release)

Cryptography

K. Bonawitz, et al. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** *CCS 2017.*



A **cryptographic protocol** that allows the server to **only see the sum** (or average) of client updates — **never the individual updates**.

1.Each client encrypts its update using a special masking technique.

2.Clients' masks cancel out when summed together.

3.The server **only sees the aggregated result**, not any single client's raw update.

How we compute

What we compute (release)

Example

Suppose 3 clients have updates: 2, 3, and 5.

•Each client adds a random mask (e.g., +7, -4, -3) and sends masked updates:

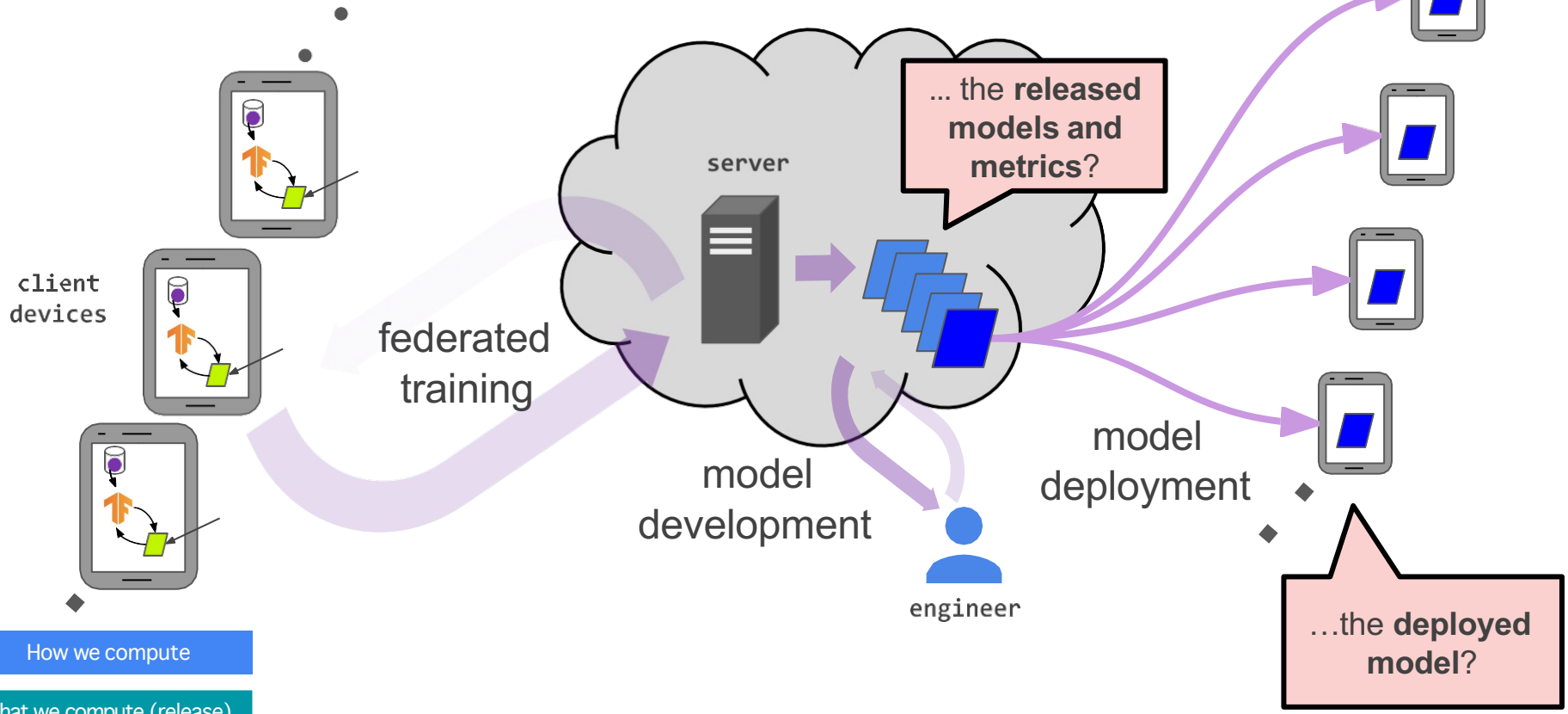
- Client 1 sends 2 + 7 = 9
- Client 2 sends 3 - 4 = -1
- Client 3 sends 5 - 3 = 2

•Server receives: 9, -1, 2 → sum is **10**.

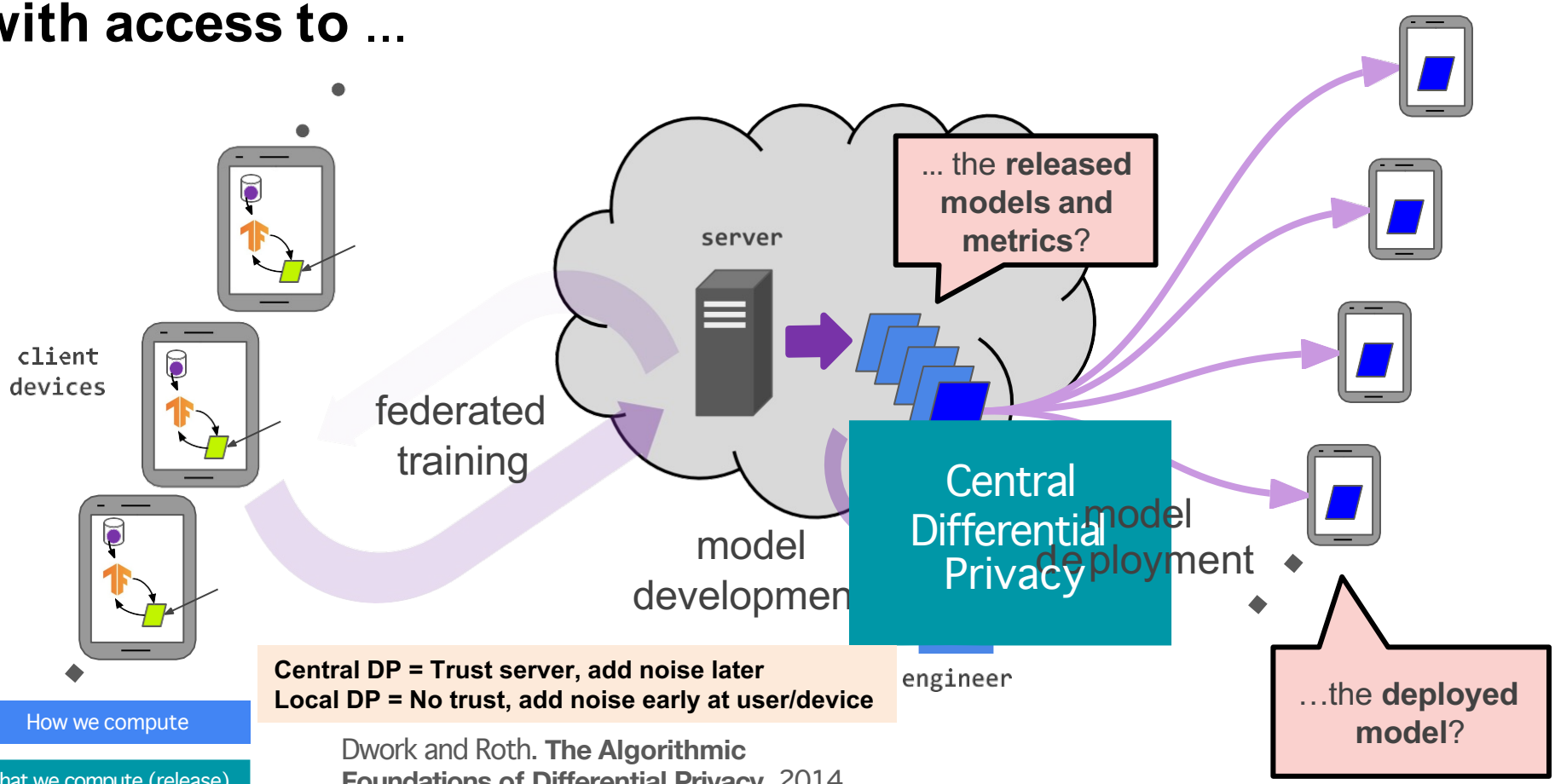
•Final unmasked sum = **10**, which matches 2+3+5.

Server **only knows 10**, **not** 2, 3, or 5.

What private information might an actor learn with access to ...



What private information might an actor learn with access to ...



Central DP = Trust server, add noise later
Local DP = No trust, add noise early at user/device

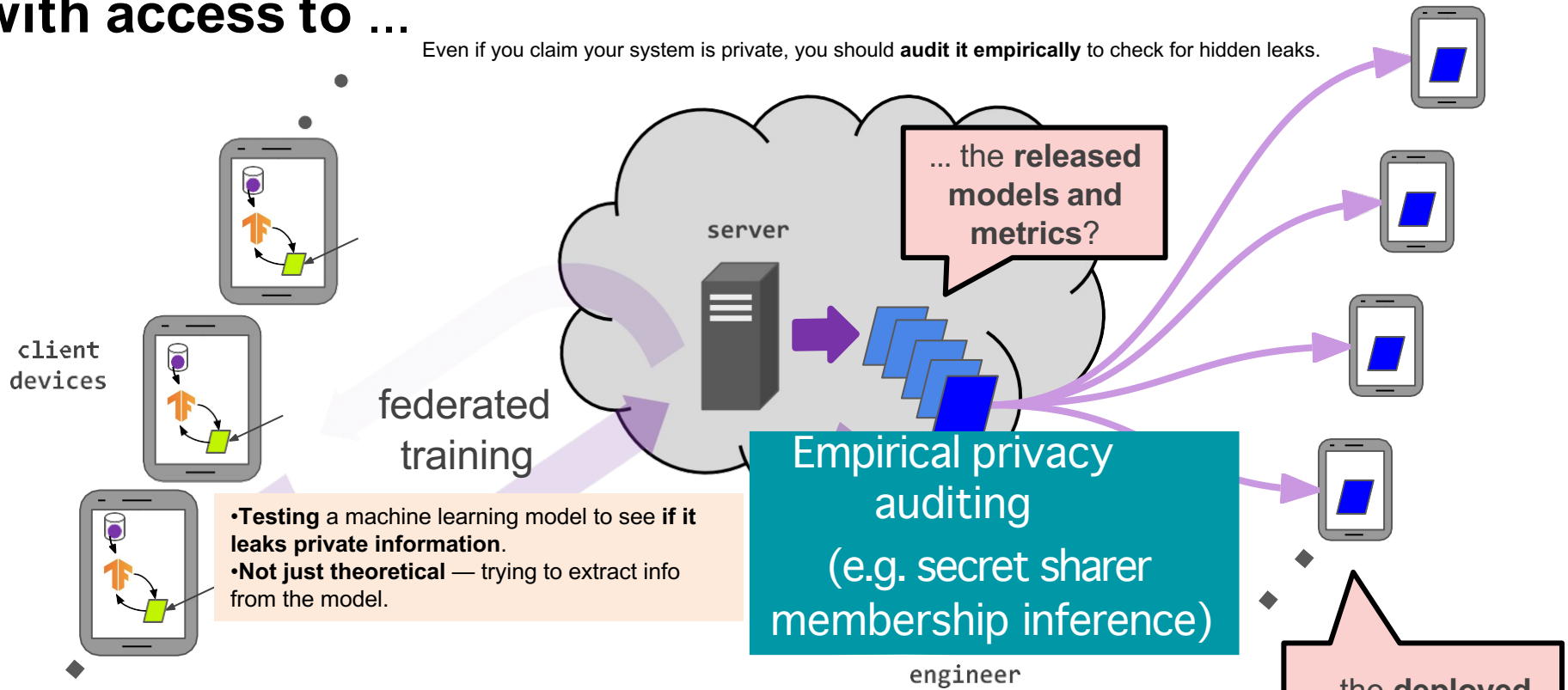
How we compute

What we compute (release)

Dwork and Roth. **The Algorithmic Foundations of Differential Privacy**. 2014.

What private information might an actor learn with access to ...

Even if you claim your system is private, you should **audit it empirically** to check for hidden leaks.



- **Testing** a machine learning model to see **if it leaks private information**.
- **Not just theoretical** — trying to extract info from the model.

How we compute

What we compute (release)

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, Dawn Song.
The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks. 2018.
Congzheng Song, Vitaly Shmatikov. **Auditing Data Provenance in Text-Generation Models.** 2018.
Matthew Jagielski, Jonathan Ullman, Alina Oprea. **Auditing Differentially Private Machine Learning: How Private is Private SGD?** 2020.

Differentially Private Federated Training

Differential Privacy

Differential privacy is the statistical science of trying to learn **as much as possible about a group** while learning **as little as possible about any individual in it.**

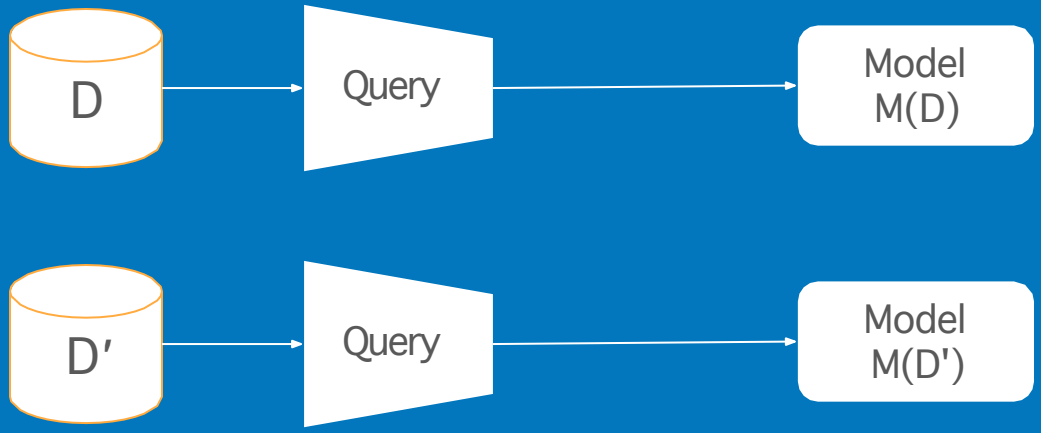
Andy Greenberg
Wired 2016.06.13

Differential Privacy

D and D′:
•Two **adjacent datasets** (differ by just **one person**'s data).

M(D) and M(D′):
•Outputs (models) trained on D and D′.

S: A set of possible model outputs



(ε, δ)-Differential Privacy: The distribution of the output $M(D)$ (a trained model) on database (training dataset) D is **nearly the same** as $M(D')$ for all adjacent databases D and D'

$$\forall S: \quad \Pr[M(D) \in S] \leq \exp(\varepsilon) \cdot \Pr[M(D') \in S] + \delta$$

Adding or removing one person's data does not significantly change the model's behavior.

(ε, δ)-Differential Privacy: The distribution of the output $M(D)$ (a trained model) on database (training dataset) D is **nearly the same** as $M(D')$ for all adjacent databases *D* and *D′*

$$\Pr[M(D) \in S] \leq e^{\varepsilon} \times \Pr[M(D') \in S] + \delta$$

If ε is **small**:

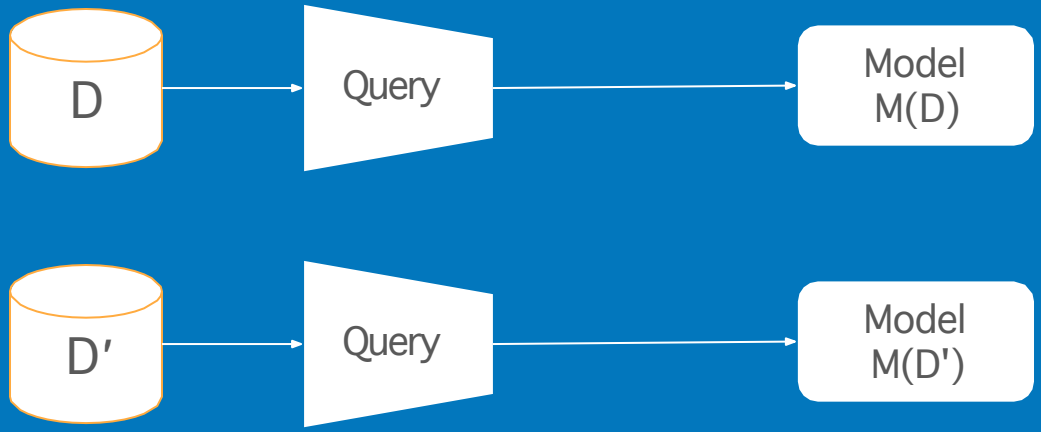
- $e^{\varepsilon} \approx 1 + \varepsilon$

If ε = 0:

- Then $e^0 = 1$.
- Perfect privacy: output distributions are identical.

- The probability that $M(D)$ gives an output in S is **almost the same** as the probability that $M(D')$ does.
- The difference is controlled by:
  - ε → **privacy loss budget** (smaller = stronger privacy)
  - δ → **small probability of failure** (slight relaxation)

Record-level Differential Privacy

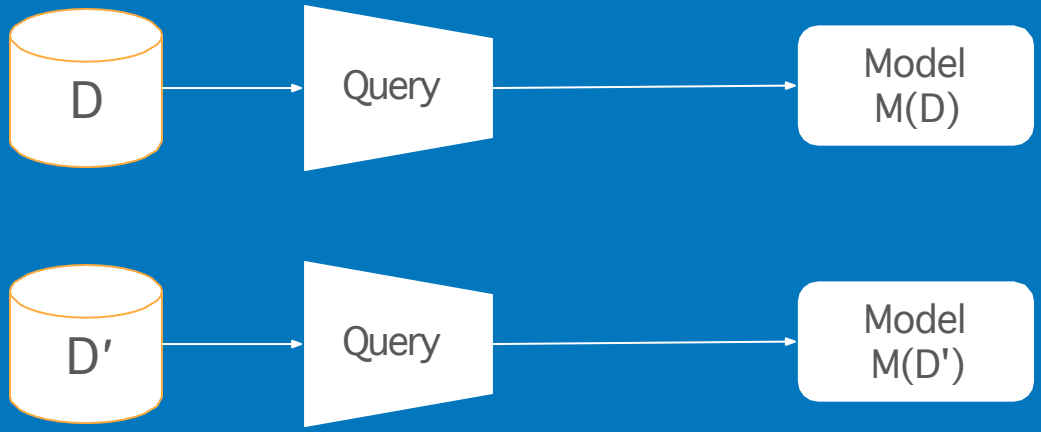


(ε, δ)-Differential Privacy: The distribution of the output $M(D)$ (a trained model) on database (training dataset) D is nearly the same as $M(D')$ for all **adjacent** databases D and D'

adjacent: Sets D and D' differ only by presence/absence of one **example**

M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, & L. Zhang. ***Deep Learning with Differential Privacy.*** *CCS 2016.*

User-level Differential Privacy



(ε, δ)-Differential Privacy: The distribution of the output $M(D)$ (a trained model) on database (training dataset) D is nearly the same as $M(D')$ for all **adjacent** databases D and D'

adjacent: Sets D and D' differ only by presence/absence of one ~~**example**~~ **user**

H. B. McMahan, et al. ***Learning Differentially Private Recurrent Language Models****. ICLR 2018.*

Scenario:
•Publish average age of patients diagnosed with disease X.

Without Differential Privacy:
•Dataset D: Ages = [20, 22, 24, 26, 80] → Average = 34.4
•Remove the 80-year-old:
Dataset D′: Ages = [20, 22, 24, 26] → New Average = 23.0

Big change → Privacy leak!

With Differential Privacy:
•Add random noise before publishing:

- D: 34.4 + 3 = 37.4
- D′: 23.0 + 15 = 38.0

Released numbers are close → Hard to tell if the 80-year-old was included.

Differential Privacy blurs individual impact by adding noise to protect privacy.
We sacrifice a little bit of accuracy to gain a lot of privacy.

Typical values of ε (epsilon) and δ (delta):

$$\Pr[M(D) \in S] \leq e^{\varepsilon} \times \Pr[M(D') \in S] + \delta$$

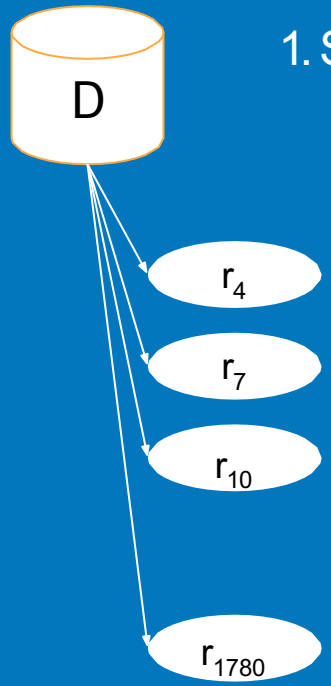
The output of M should **not depend heavily** on any one person's data — adding/removing them **barely changes** the probability of any output.

Differential Privacy = Outputs are almost indistinguishable whether any single individual is included or not.

Setting	Epsilon (ε)	Delta (δ)
Very strong privacy	0.01 – 0.1	10^{-6} or smaller
Medium privacy (common in practice)	1 – 3	10^{-6} to 10^{-8}
Weak privacy	5 – 10	10^{-6} or larger

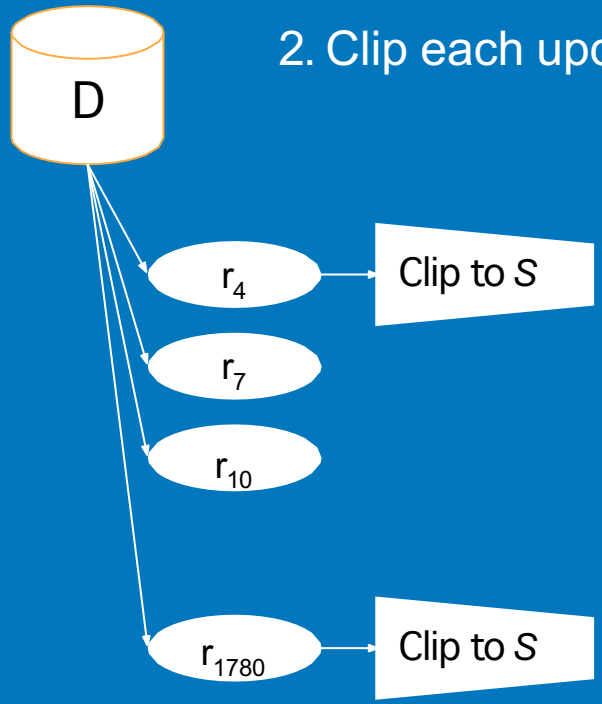
- **ε ≈ 1** is considered **good** in real-world systems.
- **δ** is usually set to something tiny, like 10^{-6}.

Iterative training with differential privacy



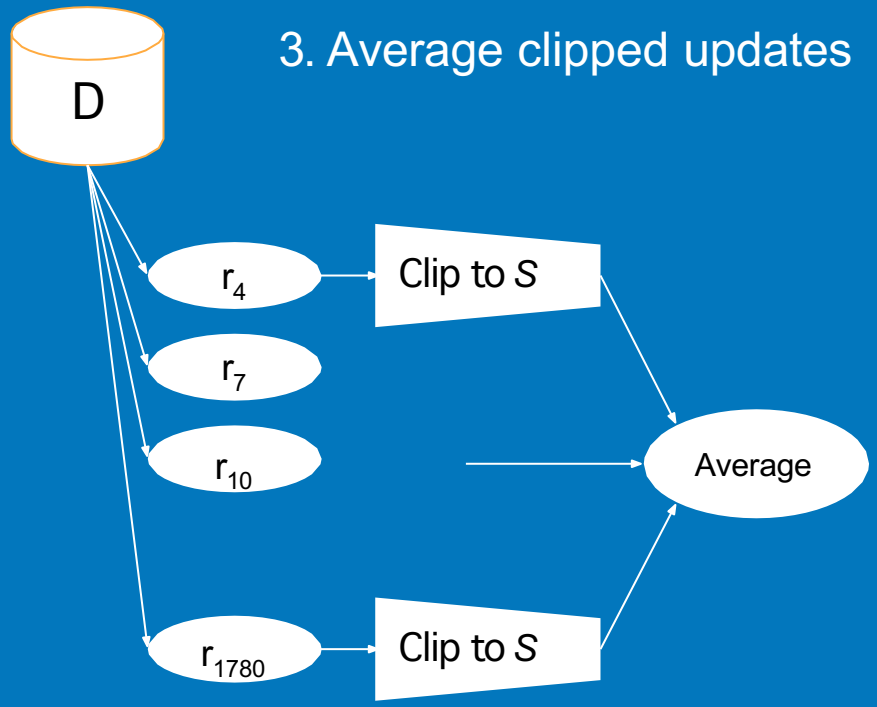
1. Sample a batch of clients uniformly at random

Iterative training with differential privacy

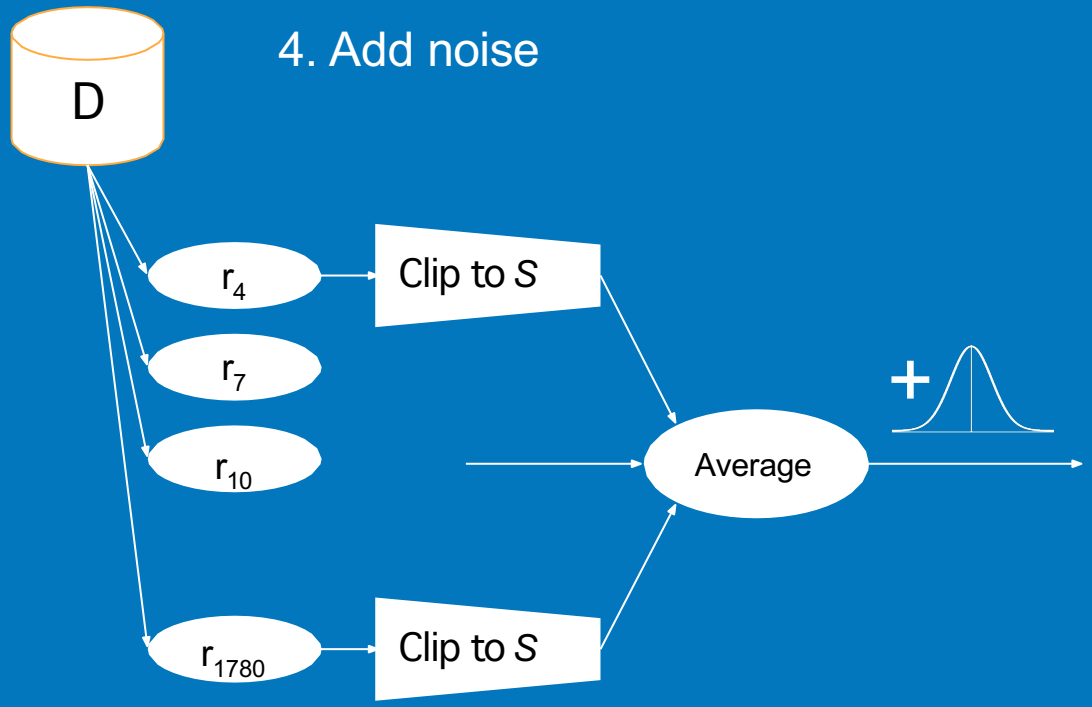


Limits how much any one data point can influence the model. Prevents extreme updates from causing big privacy leaks.

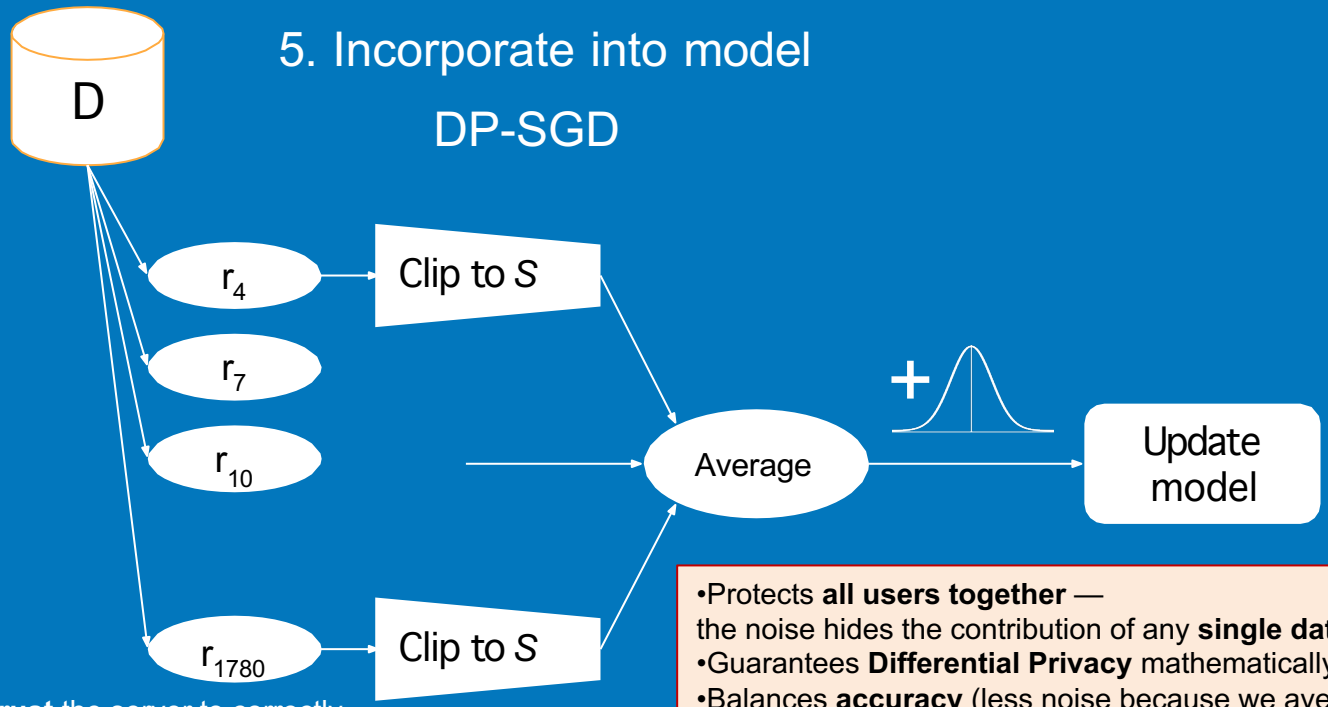
Iterative training with differential privacy



Iterative training with differential privacy



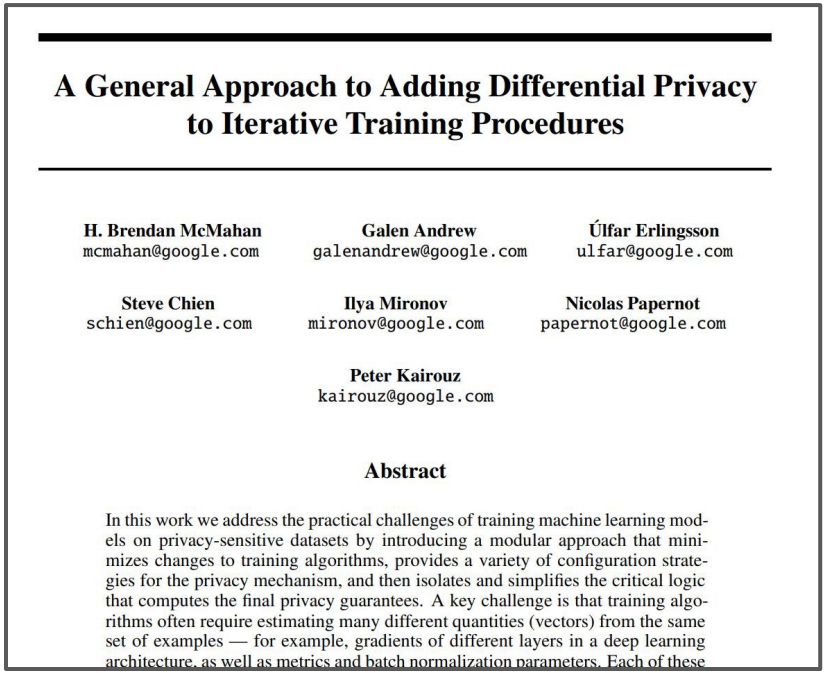
Iterative training with differential privacy



Clients trust the server to correctly clip, average, and add noise.

- Protects **all users together** — the noise hides the contribution of any **single data point**.
- Guarantees **Differential Privacy** mathematically.
- Balances **accuracy** (less noise because we average many points) and **privacy**.

There are many details and possibilities



A General Approach to Adding Differential Privacy to Iterative Training Procedures

H. Brendan McMahan
mcmahan@google.com

Galen Andrew
galenandrew@google.com

Úlfar Erlingsson
ulfar@google.com

Steve Chien
schien@google.com

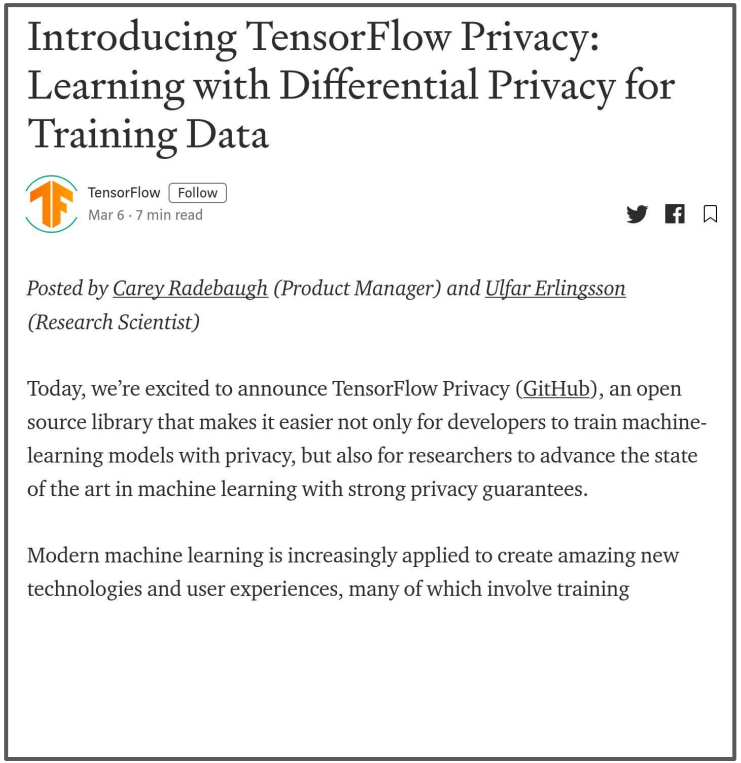
Ilya Mironov
mironov@google.com

Nicolas Papernot
papernot@google.com

Peter Kairouz
kairouz@google.com

Abstract

In this work we address the practical challenges of training machine learning models on privacy-sensitive datasets by introducing a modular approach that minimizes changes to training algorithms, provides a variety of configuration strategies for the privacy mechanism, and then isolates and simplifies the critical logic that computes the final privacy guarantees. A key challenge is that training algorithms often require estimating many different quantities (vectors) from the same set of examples — for example, gradients of different layers in a deep learning architecture, as well as metrics and batch normalization parameters. Each of these



Introducing TensorFlow Privacy: Learning with Differential Privacy for Training Data

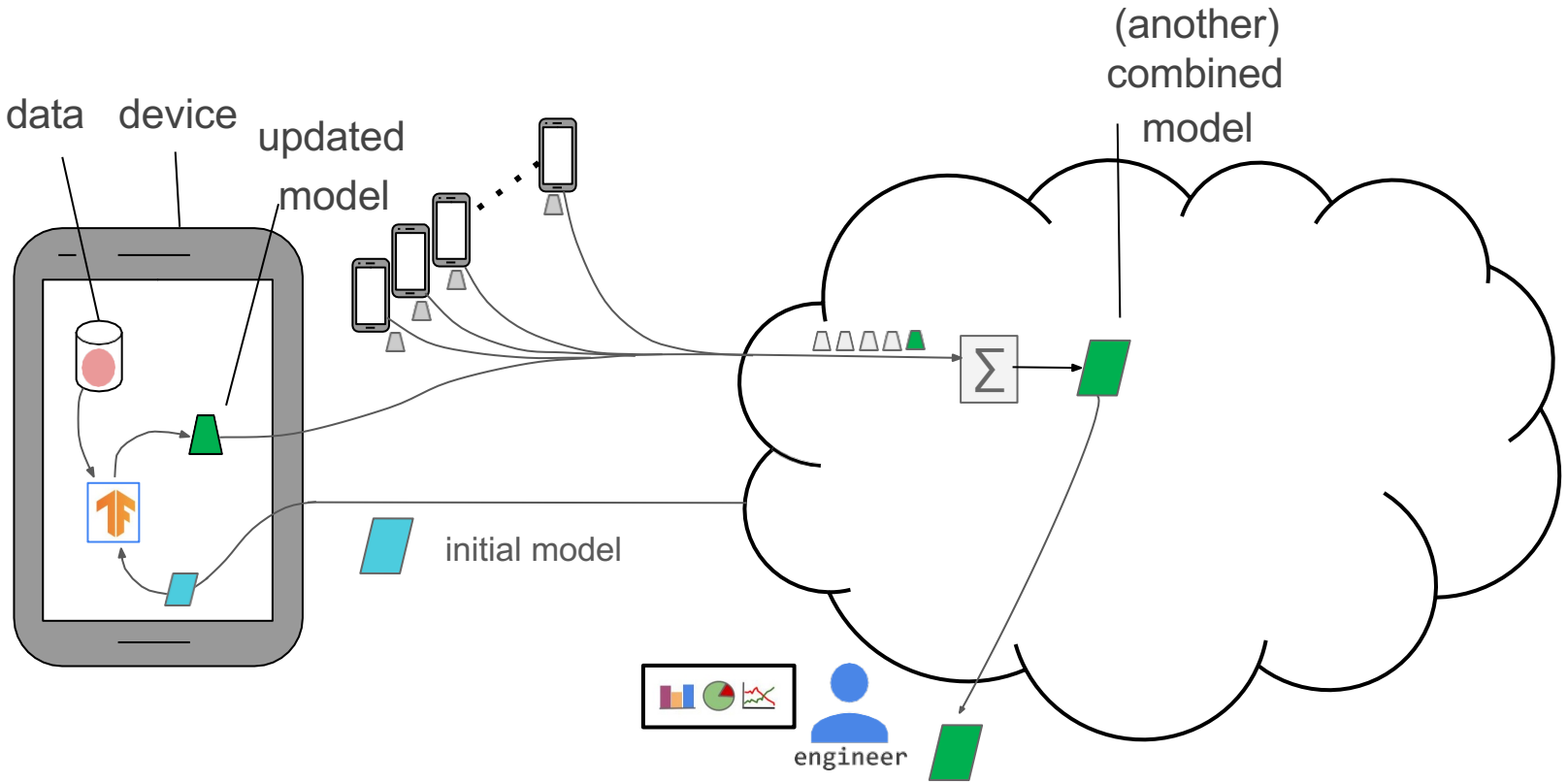
TensorFlow Follow
Mar 6 · 7 min read

Posted by Carey Radebaugh (Product Manager) and Ulfar Erlingsson (Research Scientist)

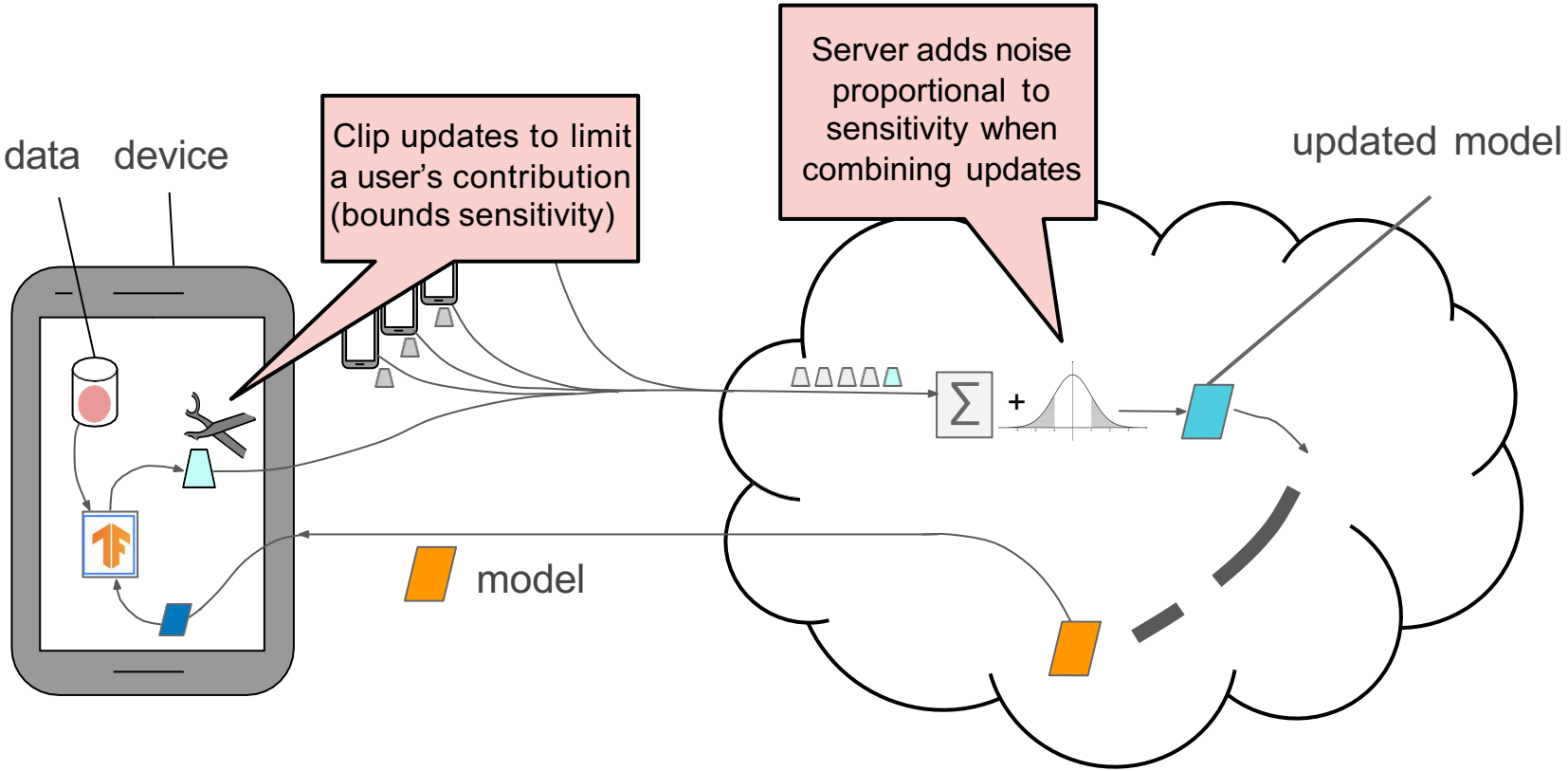
Today, we're excited to announce TensorFlow Privacy (GitHub), an open source library that makes it easier not only for developers to train machine-learning models with privacy, but also for researchers to advance the state of the art in machine learning with strong privacy guarantees.

Modern machine learning is increasingly applied to create amazing new technologies and user experiences, many of which involve training

Back to federated learning

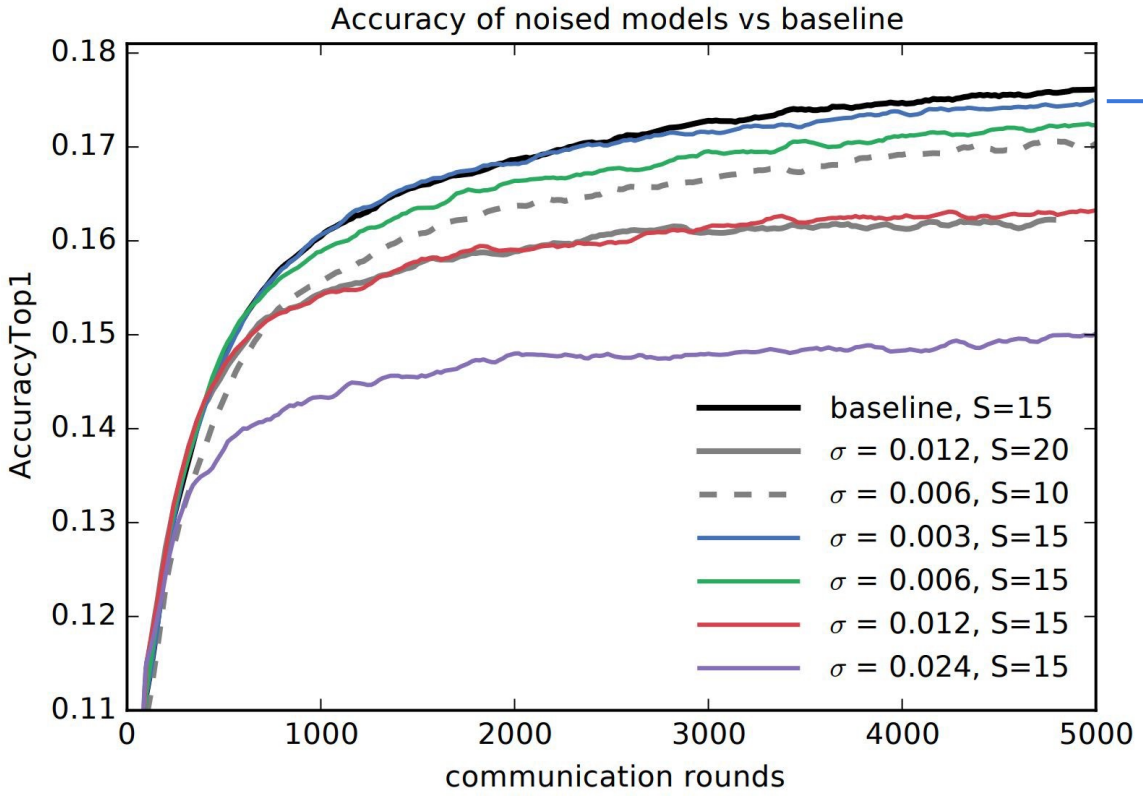


Differentially private federated learning



Differential privacy for language models

LSTM-based predictive language model.
10K word dictionary, word embeddings $\in \mathbb{R}^{96}$, state $\in \mathbb{R}^{256}$, parameters: 1.35M. Corpus=Reddit posts, by author.



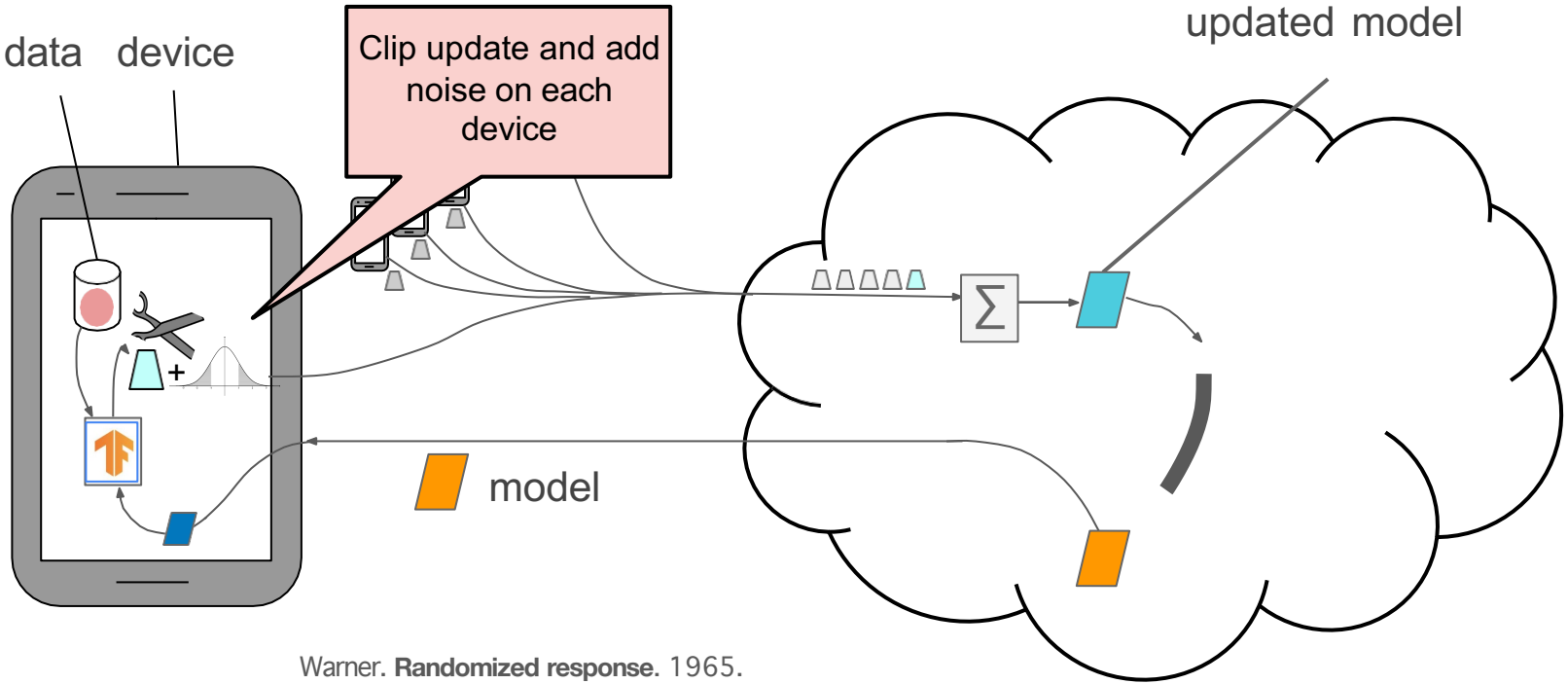
(4.634, 1e-9)-DP with 763k users
(1.152, 1e-9)-DP with 1e8 users

$\mathbb{E}$[users per minibatch] = 5k
$\mathbb{E}$[tokens per minibatch] = 8m

S: Clipping threshold
•Adding **small noise** (e.g., σ = 0.006) gives **little drop** in accuracy.
•**Larger noise** (e.g., σ = 0.024) **hurts** accuracy significantly.
•**More users (batch size)** or **better clipping** help maintain high accuracy even with DP.

H. B. McMahan, *et al*. **Learning Differentially Private Recurrent Language Models.** *ICLR 2018.*

Locally differentially private federated learning



Warner. **Randomized response.** 1965.
Kasiviswanathan, et. al. **What can we learn privately?** 2011.

Central DP:

easier to get high utility with good privacy

Local DP:

requires much weaker trust assumptions

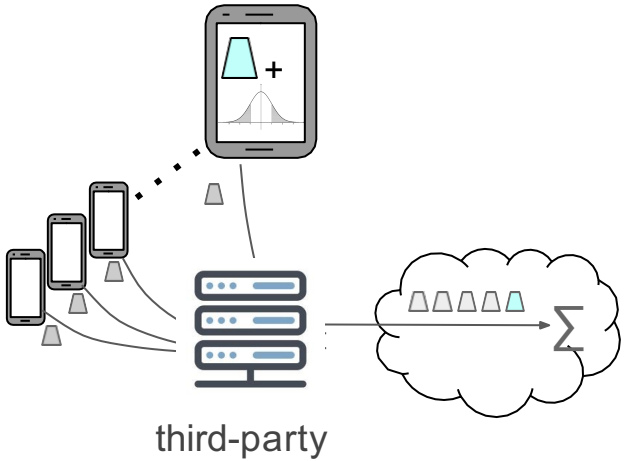
Can we combine the best of both worlds?

Distributed Differential Privacy

Distributed trust models of DP

Distributed trust models of DP

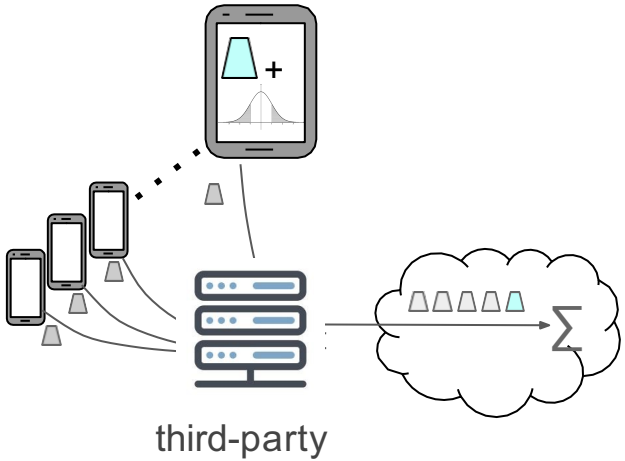
1 Trusted “third party”



•A third-party server collects all the data.
•It **adds noise itself** and **aggregates** privately.
•Users must **fully trust** the third party to behave correctly.

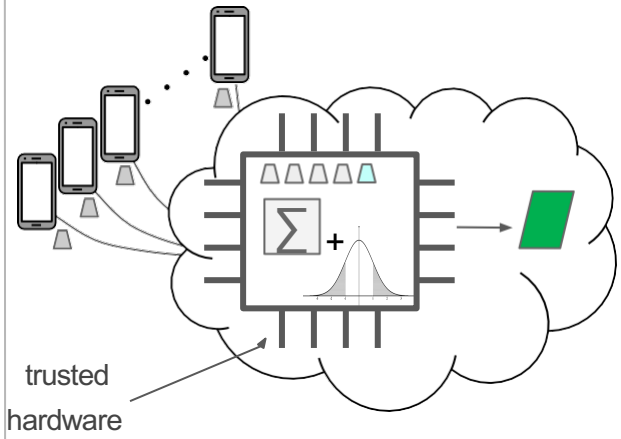
Distributed trust models of DP

1 Trusted “third party”



- A third-party server collects all the data.
- It **adds noise itself** and **aggregates** privately.
- Users must **fully trust** the third party to behave correctly.

2 Trusted Execution Environments

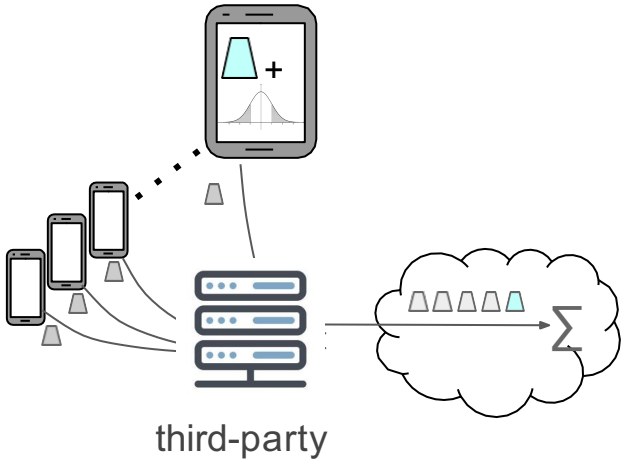


- Data goes to a **secure hardware enclave** (like Intel SGX).
- The enclave safely **aggregates** and **adds noise**.
- Even if the rest of the server is hacked, the enclave keeps the data safe.

Distributed trust models of DP

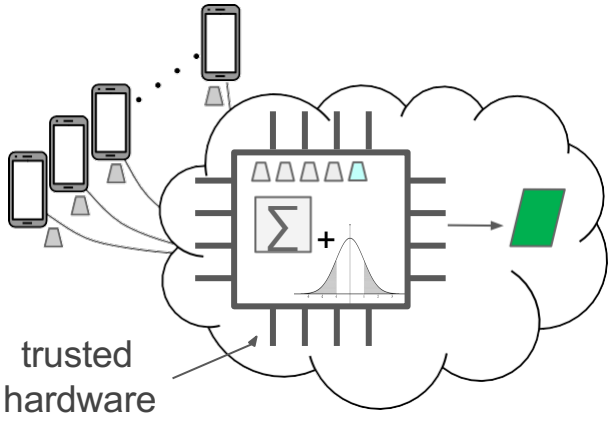
No single point of trust — trust is distributed cryptographically.

1 Trusted "third party"



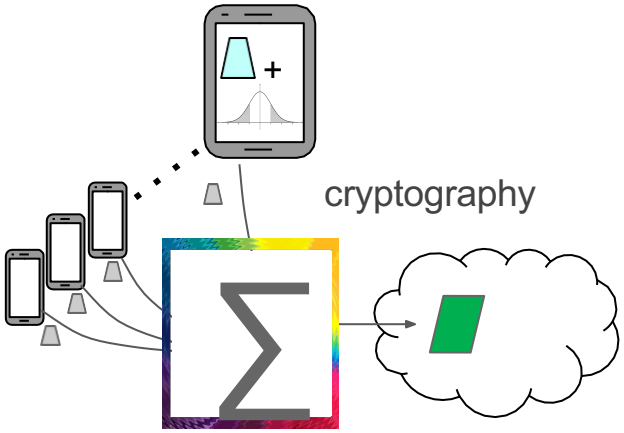
- A third-party server collects all the data.
- It **adds noise itself** and **aggregates** privately.
- Users must **fully trust** the third party to behave correctly.

2 Trusted Execution Environments



- Data goes to a **secure hardware enclave** (like Intel SGX).
- The enclave safely **aggregates** and **adds noise**.
- Even if the rest of the server is hacked, the enclave keeps the data safe.

3 Trust via Cryptography

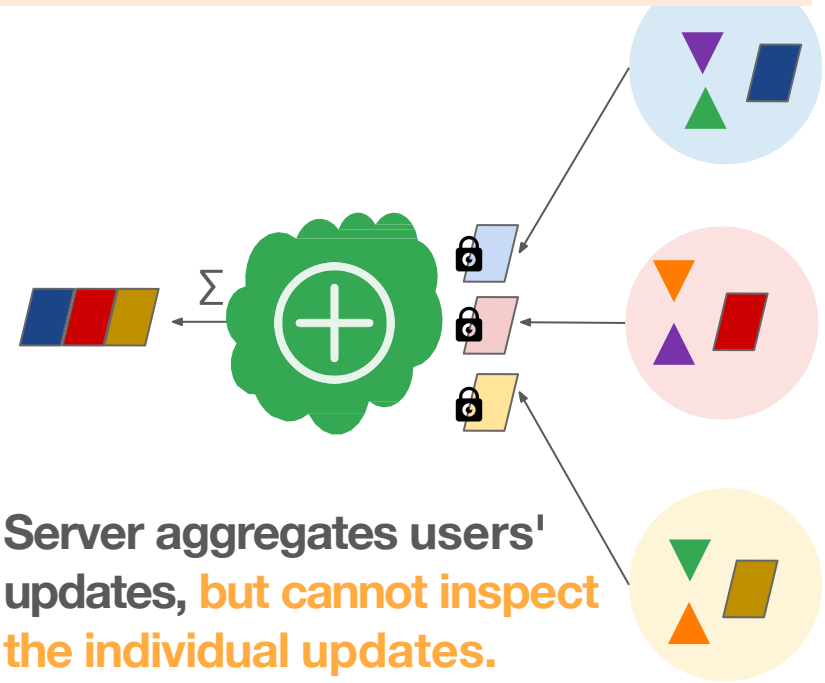


- Clients **encrypt** their updates.
- A cryptographic protocol (e.g., secure multi-party computation) ensures:
  - Only the **aggregate** can be seen
  - **No individual update** is ever revealed.
- Noise is **added either before or during aggregation**.

Secure Aggregation

•Each user's update is **masked** using cryptography.
•Only the **sum** (aggregate) is revealed after decryption.

Server learns the **total**, not **who contributed what**.



Server aggregates users' updates, but cannot inspect the individual updates.

K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** *CCS'17.*

Interactive Cryptographic Protocol

Each phase, 1000 clients + server interchange messages over 4 rounds of communication.

Secure

⅓ malicious clients + fully observed server

Robust

⅓ clients can drop out

Protects privacy even if ⅓ of clients are malicious or Server is fully observed (e.g., hacked).

Communication Efficient

# Params	Bits/Param	# Users	Expansion
2^{20} = 1 m	16	2^{10} = 1 k	**1.73x**
2^{24} = 16 m	16	2^{14} = 16 k	**1.98x**

Slight overhead (~1.7x to 2x expansion) even for large models: 1 million to 16 million parameters.

Secure Aggregation

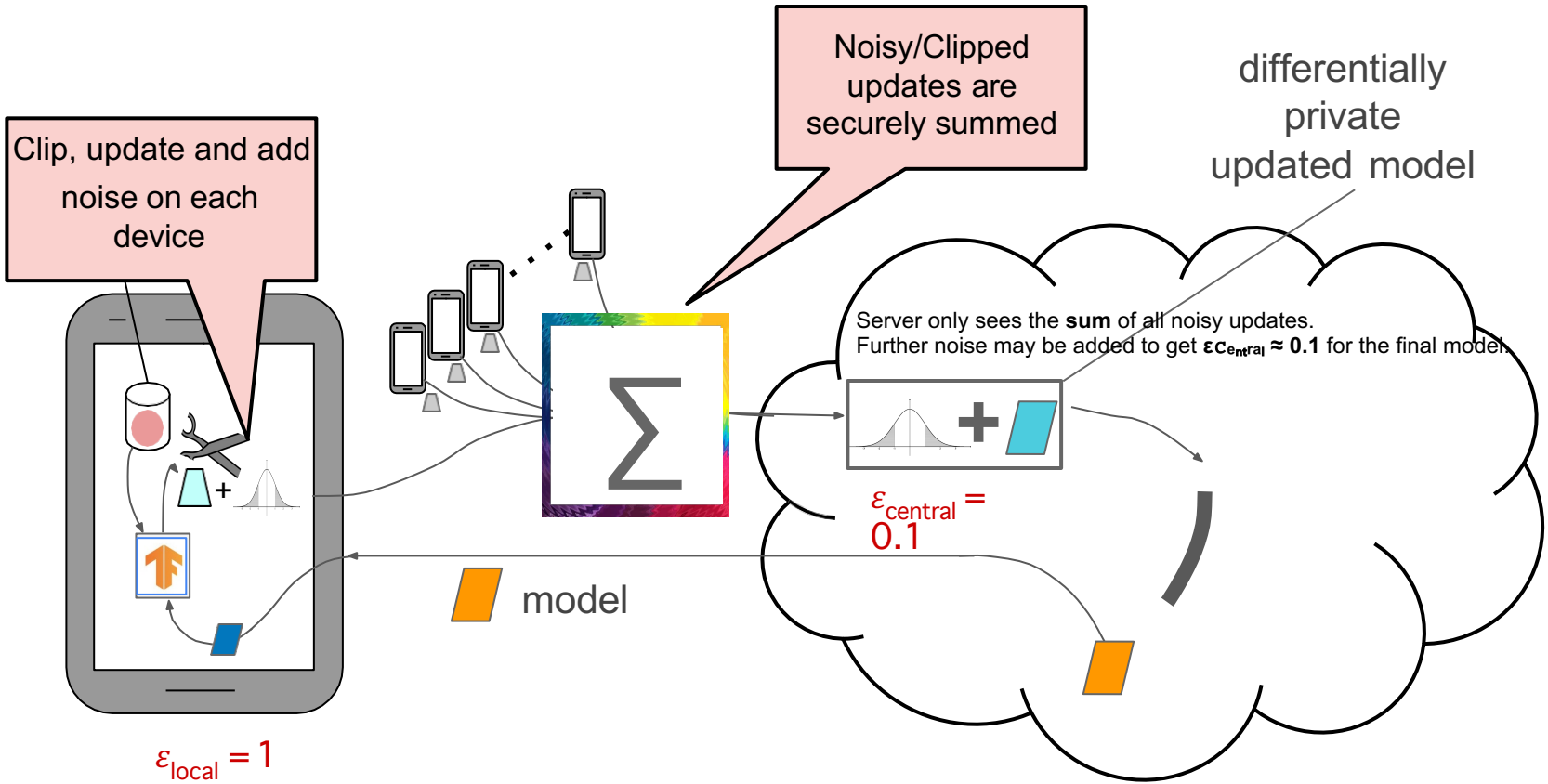
K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** *CCS 2017.*

J. Bell, K. Bonawitz, A. Gascon, T. Lepoint, M. Raykova ***Secure Single-Server Aggregation with (Poly)Logarithmic Overhead.*** CCS 2020.

Protocol	Server		Client	
	Computation	Communication	Computation	Communication
Bonawitz et al. (CCS 2017)	$O(n^2 l)$	$O(n^2 + nl)$	$O(n^2 + nl)$	$O(n + l)$
Bell et al. (CCS 2020)	$O(n \log^2 n + nl \log n)$	$O(n \log n + nl)$	$O(\log^2 n + l \log n)$	$O(\log n + l)$
Insecure Solution	$O(nl)$	$O(nl)$	$O(l)$	$O(l)$

- **n** = number of users (clients)
- **l** = number of model parameters (dimensions of each client's update)

Distributed DP via secure aggregation



Strong local protection + strong global privacy using secure aggregation and distributed noise addition.

Precise DP Guarantees for Real-World Cross-Device FL

Challenges

- There is no fixed or known database / dataset / population size
- Client availability is dynamic due to multiple system layers and participation constraints
    - "Sample from the population" or "shuffle devices" don't work out-of-the-box
- Clients may drop out at any point of the protocol, with possible impacts on privacy and utility

For **privacy** purposes, model the environment (availability, dropout) as the choices of *Nature* (possibly malicious and adaptive to previous mechanism choices)

In distributed DP, users' unpredictability must be treated as part of the threat model — not just a technical glitch.

Goals

- **Robust** to Nature's choices (client availability, client dropout) in that privacy and utility are both preserved, possibly at the expense of forward progress.
- **Self-accounting**, in that the server can compute a precise upper bound on the (ε,δ) of the mechanism using only information available via the protocol.
- During training, the server itself can track and compute
- the privacy loss (ε, δ) based on observable events only
- **Local selection**, so most participation decisions are made locally, and as few devices as possible check-in to the server
- **Good privacy vs. utility tradeoffs**

Algorithm 2 Protocol schema for DP in Cross-Device FL

$\theta_0 =$ (initialization)

for each protocol step $t = 1, 2, \ldots$ **do**

Nature (maybe malicious, adaptive) chooses $C^{\text{AVAILABLE}} \subseteq C^{\text{POPULTATION}}$

\# Clients in $C_t^{\text{AVAILABLE}}$ decide locally whether to check in to the server

$C_t^{\text{CHECKEDIN}} = \{u \mid \texttt{ShouldCheckIn}(u, t, \ldots) = 1, u \in C_t^{\text{AVAILABLE}}\}$

$C_t^{\text{SELECTED}} = \texttt{SelectFrom}(C_t^{\text{CHECKEDIN}}, \ldots)$

Nature chooses the clients $C_t^{\text{REPORTED}} \subseteq C_t^{\text{SELECTED}}$ that report

$X_t = \texttt{Aggregate}(\{\texttt{LocalUpdate}(u, \theta_t) \mid u \in C_t^{\text{REPORTED}}\})$

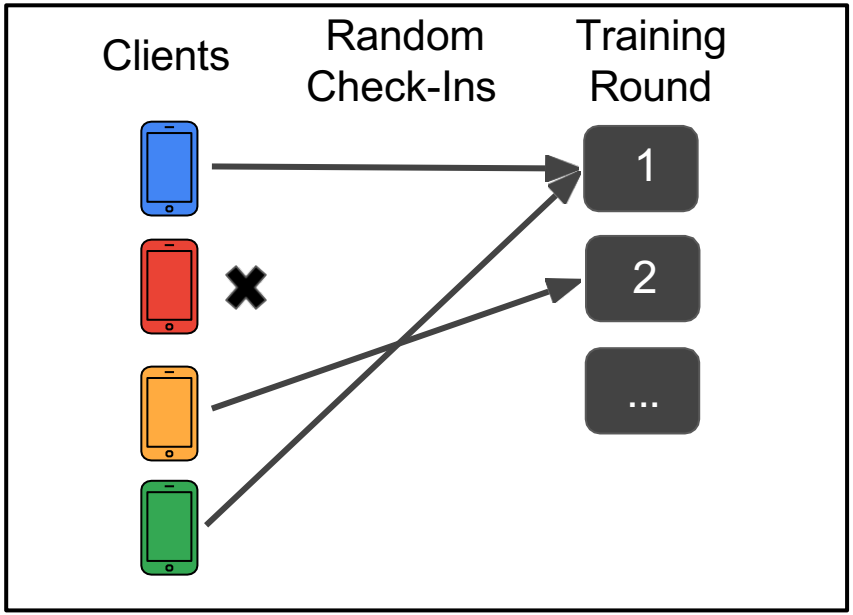
\# DP should allow the release of X_t

$\theta_{t+1} = \texttt{ServerUpdate}(\theta_t, X_t)$

Server outputs θ_{t+1} and (ϵ, δ)

`ShouldCheckIn`	Server	Runs locally on client, decides to connect to server
`SelectFrom`	Client	Selects devices to participate from CHECKEDIN
`LocalUpdate`	Client	Computes the value to report to the server
`Aggregate`	Server	Aggregates updates to produce DP output
`ServerUpdate`	Server	Update server state (DP post processing)

Random Check-ins



Randomness in who shows up = stronger privacy for free!

Privacy Amplification via Random Check-Ins

Borja Balle* Peter Kairouz† H. Brendan McMahan† Om Thakkar†
Abhradeep Thakurta‡

July 15, 2020

Abstract

Differentially Private Stochastic Gradient Descent (DP-SGD) forms a fundamental building block in many applications for learning over sensitive data. Two standard approaches, privacy amplification by subsampling, and privacy amplification by shuffling, permit adding lower noise in DP-SGD than via naïve schemes. A key assumption in both these approaches is that the elements in the data set can be uniformly sampled, or be uniformly permuted — constraints that may become prohibitive when the data is processed in a decentralized or distributed fashion. In this paper, we focus on conducting iterative methods like DP-SGD in the setting of federated learning (FL) wherein the data is distributed among many devices (clients). Our main contribution is the *random check-in* distributed protocol, which crucially relies only on randomized participation decisions made locally and independently by each client. It has privacy/accuracy trade-offs similar to privacy amplification by subsampling/shuffling. However, our method does not require server-initiated communication, or even knowledge of the population size. To our knowledge, this is the first privacy amplification tailored for a distributed learning framework, and it may have broader applicability beyond FL. Along the way, we extend privacy amplification by shuffling to incorporate (ε, δ)-DP local randomizers, and exponentially improve its guarantees. In practical regimes, this improvement allows for similar privacy and utility using data from an order of magnitude fewer users.

1 Introduction

Modern mobile devices and web services benefit significantly from large-scale machine learning, often involving training on user (client) data. When such data is sensitive, steps must be taken to ensure privacy, and a formal guarantee of differential privacy (DP) [15, 16] is the gold standard. For this reason, DP has been adopted by companies including Google [9, 18, 20], Apple [2], Microsoft [13], and LinkedIn [31], as well as the US Census Bureau [26].

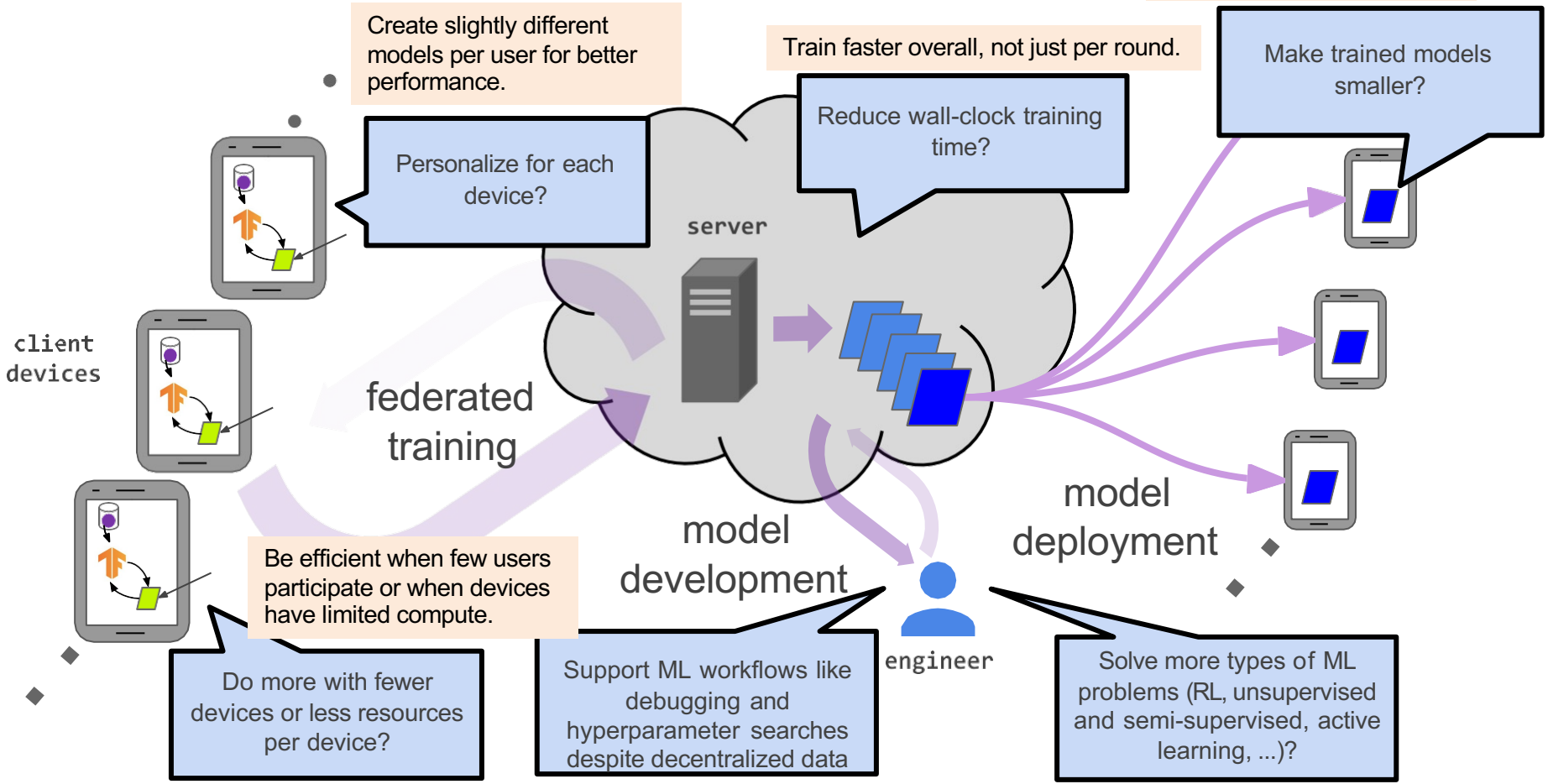
Other privacy-enhancing techniques can be combined with DP to obtain additional benefits. In particular, cross-device federated learning (FL) [27] allows model training while keeping client data decentralized (each participating device keeps its own local dataset, and only sends model updates or gradients to the coordinating server). However, existing approaches to combining FL and DP make a number of assumptions that are unrealistic in real-world FL deployments such as [10]. To highlight these challenges, we must first review the state-of-the-art in centralized DP training, where differentially private stochastic gradient descent (DP-SGD) [1, 8, 34] is ubiquitous. It achieves optimal error for convex problems [8], and can also be applied to non-convex problems, including deep learning, where the privacy amplification offered by randomly subsampling data to form batches is critical for obtaining meaningful DP guarantees [1, 5, 8, 25, 37].

Attempts to combine FL and the above lines of DP research have been made previously; notably, [3, 28] extended the approach of [1] to FL and user-level DP. However, these works and others in the area sidestep a critical issue: the DP guarantees require very specific sampling or shuffling schemes assuming, for example, that each client participates in each iteration with a fixed probability. While possible in theory, such schemes are incompatible with the practical

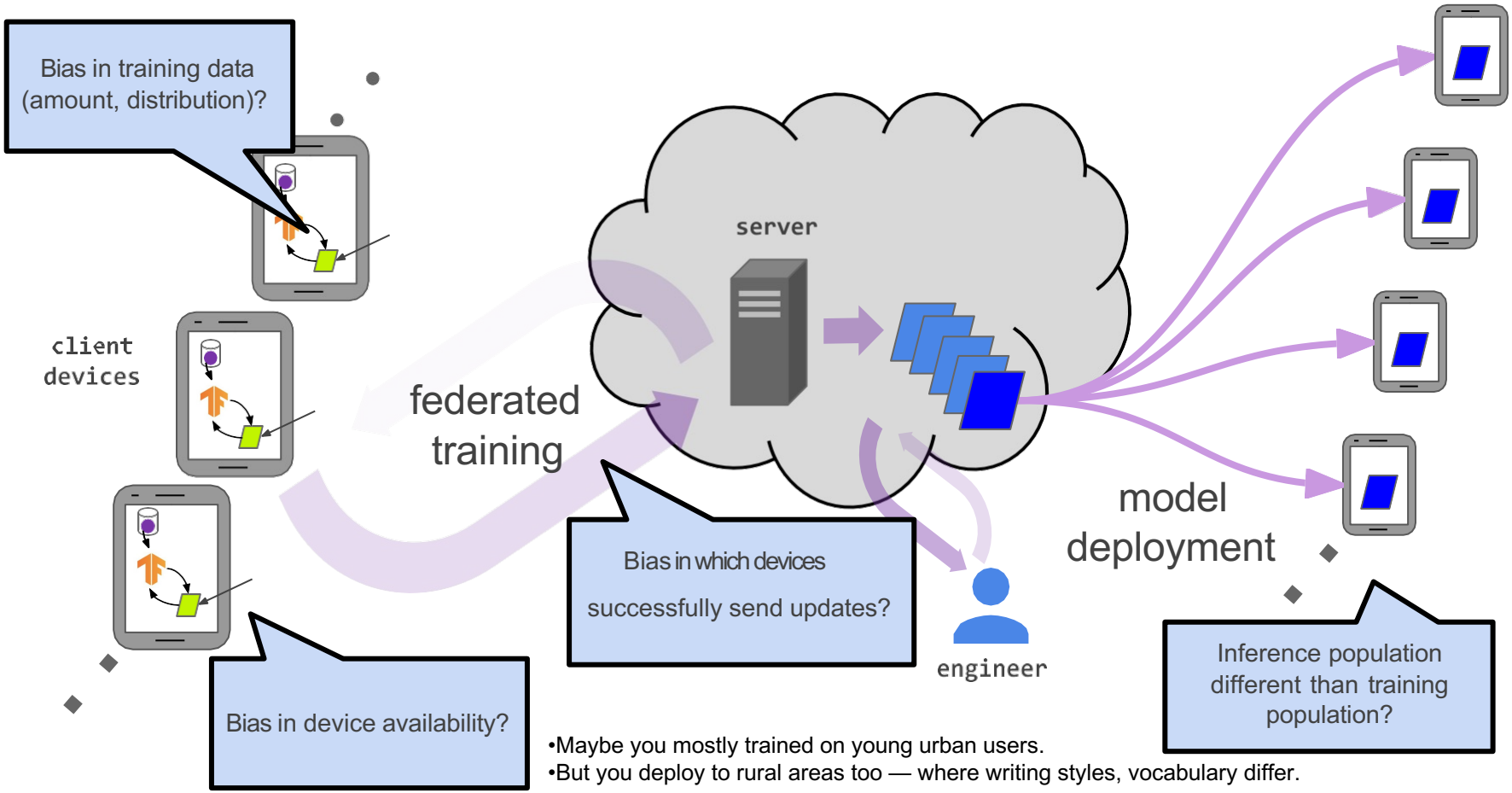
*DeepMind. bballe@google.com
†Google. {kairouz, mcmahan, omthkkr}@google.com
‡Google Research - Brain. {athakurta}@google.com

Part IV: Open Problems and Other Topics

Improving efficiency and effectiveness



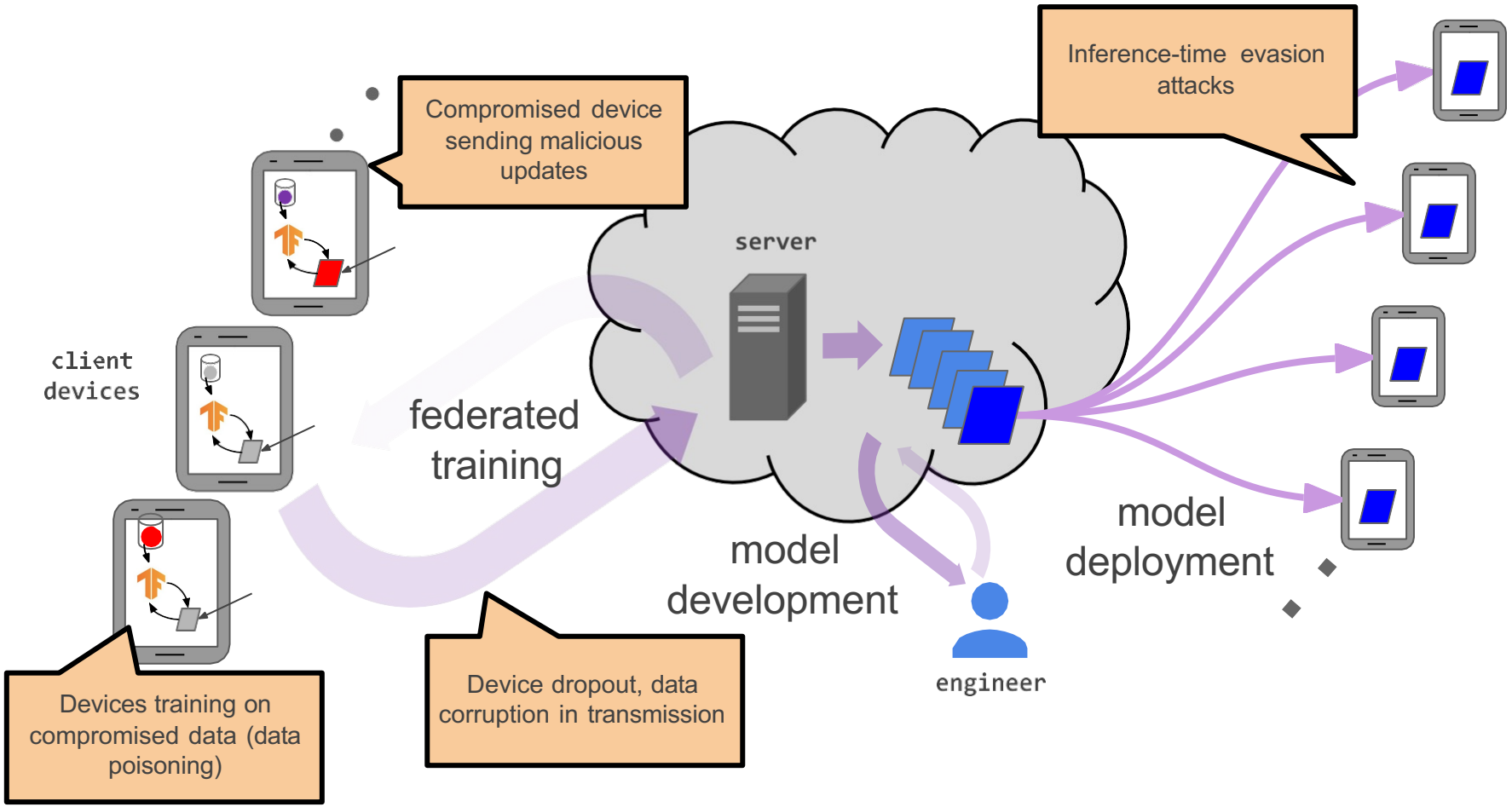
Ensuring fairness and addressing sources of bias



•Maybe you mostly trained on young urban users.
•But you deploy to rural areas too — where writing styles, vocabulary differ.

Robustness to attacks and failures

e.g. Insert tiny typos or synonyms to **trick spam detectors**.



Advances and Open Problems in Federated Learning

Peter Kairouz[7*] H. Brendan McMahan[7*] Brendan Avent[21] Aurélien Bellet[9]
Mehdi Bennis[19] Arjun Nitin Bhagoji[13] Keith Bonawitz[7] Zachary Charles[7]
Graham Cormode[23] Rachel Cummings[6] Rafael G.L. D'Oliveira[14]
Salim El Rouayheb[14] David Evans[22] Josh Gardner[24] Zachary Garrett[7]
Adrià Gascón[7] Badih Ghazi[7] Phillip B. Gibbons[2] Marco Gruteser[7,14]
Zaid Harchaoui[24] Chaoyang He[21] Lie He [4] Zhouyuan Huo [20]
Ben Hutchinson[7] Justin Hsu[25] Martin Jaggi[4] Tara Javidi[17] Gauri Joshi[2]
Mikhail Khodak[2] Jakub Konečný[7] Aleksandra Korolova[21] Farinaz Koushanfar[17]
Sanmi Koyejo[7,18] Tancrède Lepoint[7] Yang Liu[12] Prateek Mittal[13]
Mehryar Mohri[7] Richard Nock[1] Ayfer Özgür[15] Rasmus Pagh[7,10]
Mariana Raykova[7] Hang Qi[7] Daniel Ramage[7] Ramesh Raskar[11]
Dawn Song[16] Weikang Song[7] Sebastian U. Stich[4] Ziteng Sun[3]
Ananda Theertha Suresh[7] Florian Tramèr[15] Praneeth Vepakomma[11] Jianyu Wang[2]
Li Xiong[5] Zheng Xu[7] Qiang Yang[8] Felix X. Yu[7] Han Yu[12] Sen Zhao[7]

[1]Australian National University, [2]Carnegie Mellon University, [3]Cornell University,
[4]École Polytechnique Fédérale de Lausanne, [5]Emory University, [6]Georgia Institute of Technology,
[7]Google Research, [8]Hong Kong University of Science and Technology, [9]INRIA, [10]IT University of Copenhagen,
[11]Massachusetts Institute of Technology, [12]Nanyang Technological University, [13]Princeton University,
[14]Rutgers University, [15]Stanford University, [16]University of California Berkeley,
[17] University of California San Diego, [18]University of Illinois Urbana-Champaign, [19]University of Oulu,
[20]University of Pittsburgh, [21]University of Southern California, [22]University of Virginia,
[23]University of Warwick, [24]University of Washington, [25]University of Wisconsin–Madison

Abstract

Federated learning (FL) is a machine learning setting where many clients (e.g. mobile devices or whole organizations) collaboratively train a model under the orchestration of a central server (e.g. service provider), while keeping the training data decentralized. FL embodies the principles of focused data collection and minimization, and can mitigate many of the systemic privacy risks and costs resulting from traditional, centralized machine learning and data science approaches. Motivated by the explosive growth in FL research, this paper discusses recent advances and presents an extensive collection of open problems and challenges.

Advances and Open Problems in FL

58 authors from 25 top institutions

arxiv.org/abs/1912.04977

Prat IV Outline

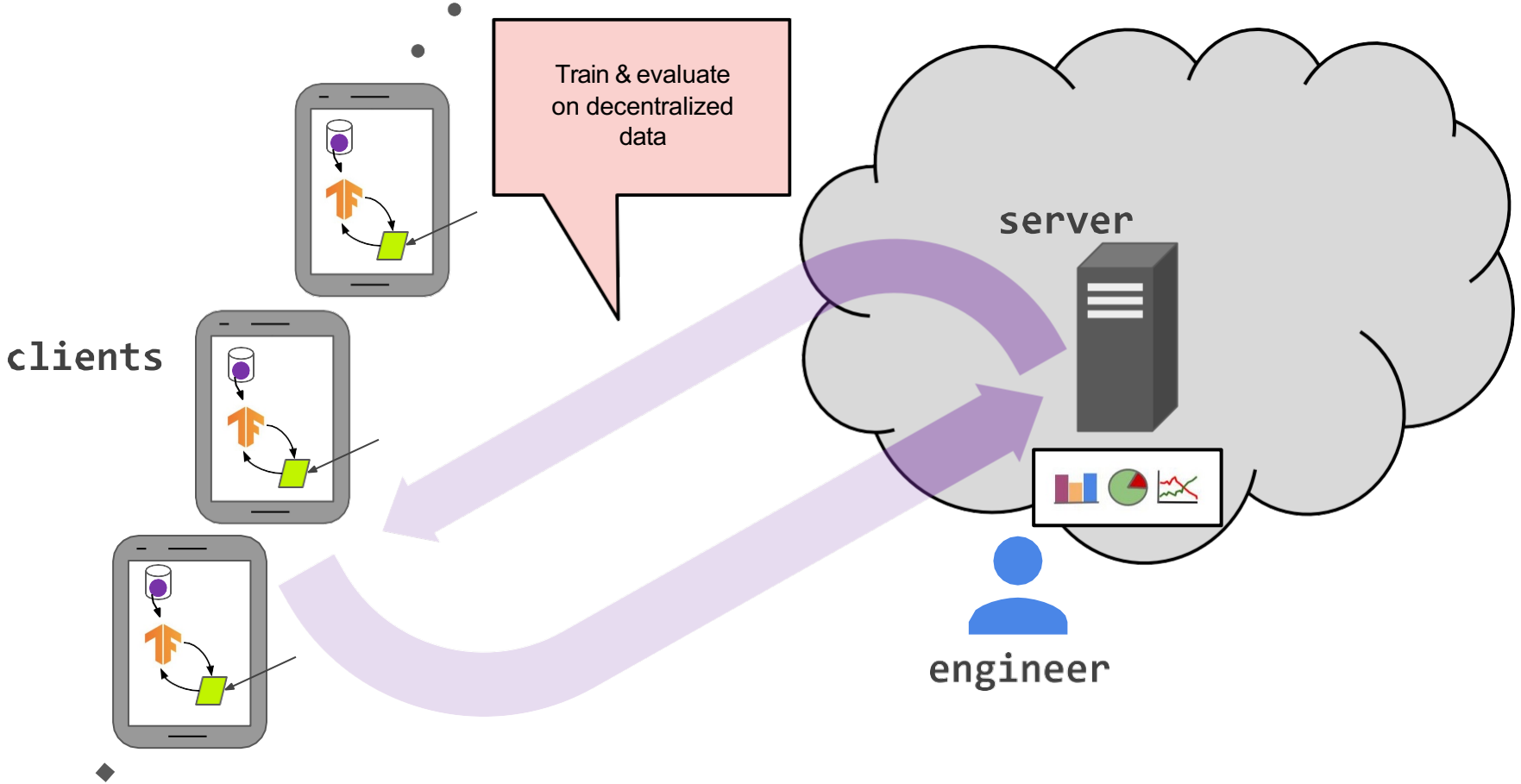
1. Robustness against failures and attacks

2. Modeling heterogeneous data: fairness & personalization

3. System challenges

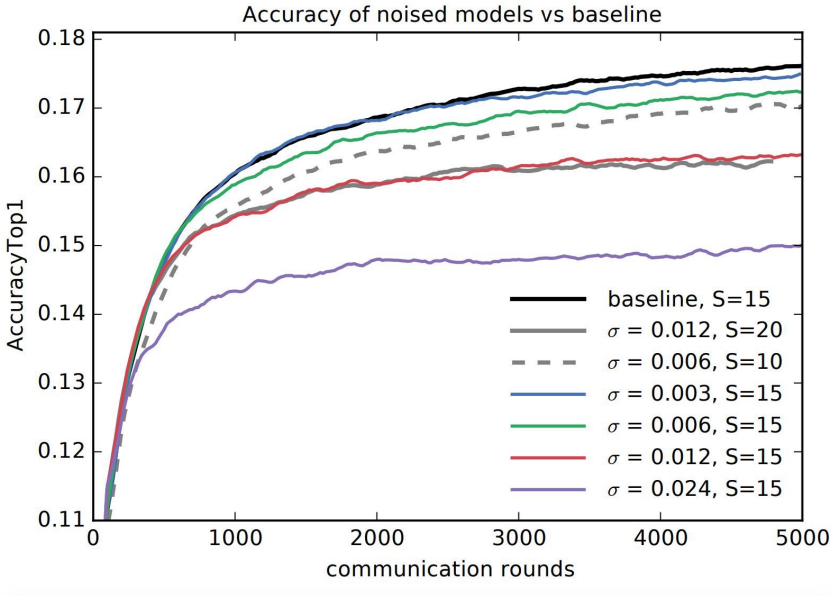
Robustness against failures and attacks

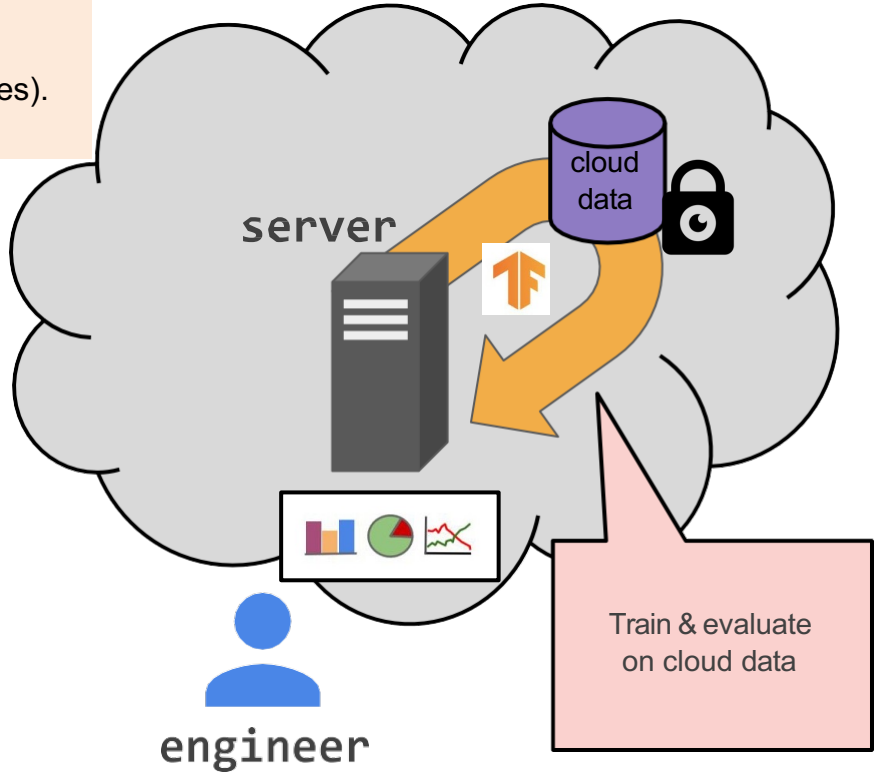
Federated training



This is what we like to see

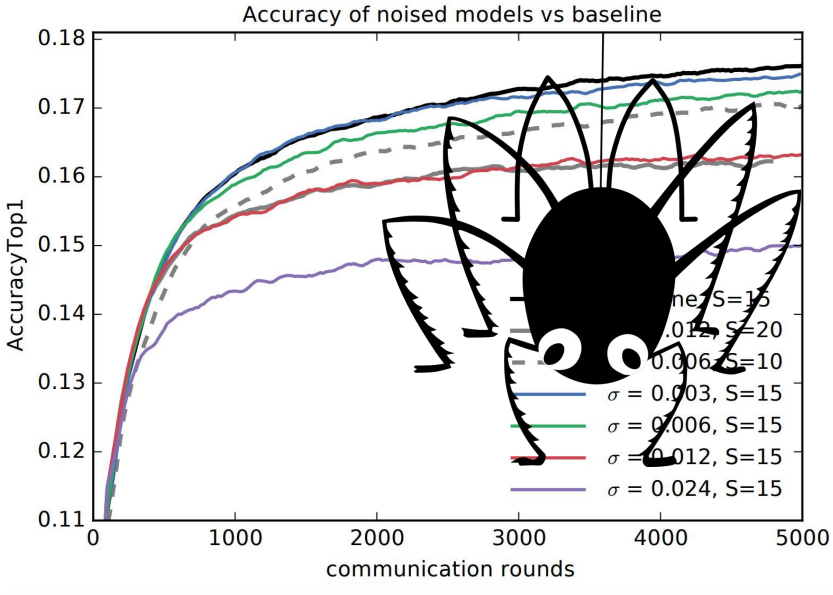
- After federated training, we **evaluate on trusted cloud data** (e.g., public datasets).
- The model shows **good, clean learning curves** (accuracy improves).
- Evaluation matches expectations.

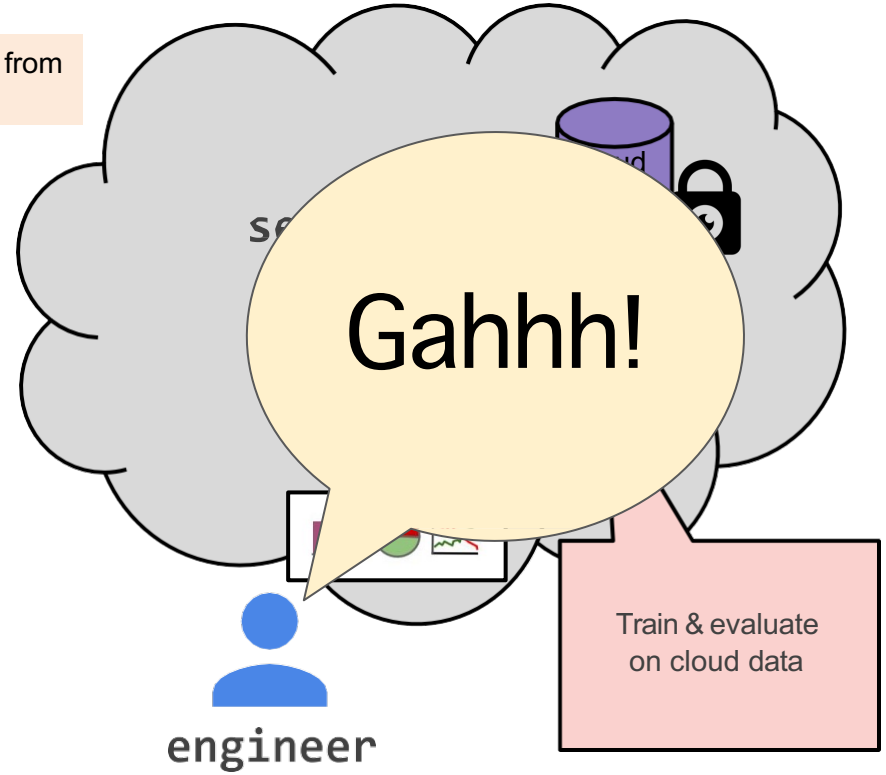




…but sometimes we see this

training data (users' real-world, noisy, biased data) was **very different** from **cloud evaluation data**





Typical ML tasks require data inspection

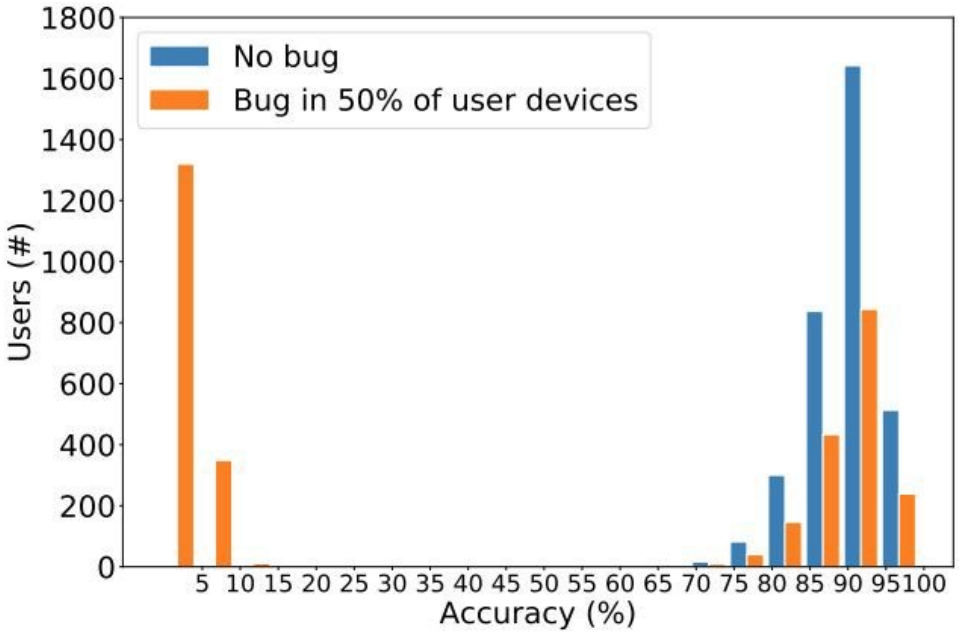
Task		Selection criteria for data to inspect
T1	Sanity checking data	Random training examples
T2	Debugging mistakes	Misclassified examples (by the primary classifier)
T3	Debugging unknown labels/classes, e.g. out-of-vocabulary words	Examples of the unknown labels/classes
T4	Debugging poor performance on certain classes/slices/users	Examples from the low-accuracy classes/slices/users
T5	Human labeling of examples	Unlabeled examples from the training distribution
T6	Detecting bias in the training data	Examples with high density in the serving distribution but low density in the training distribution.

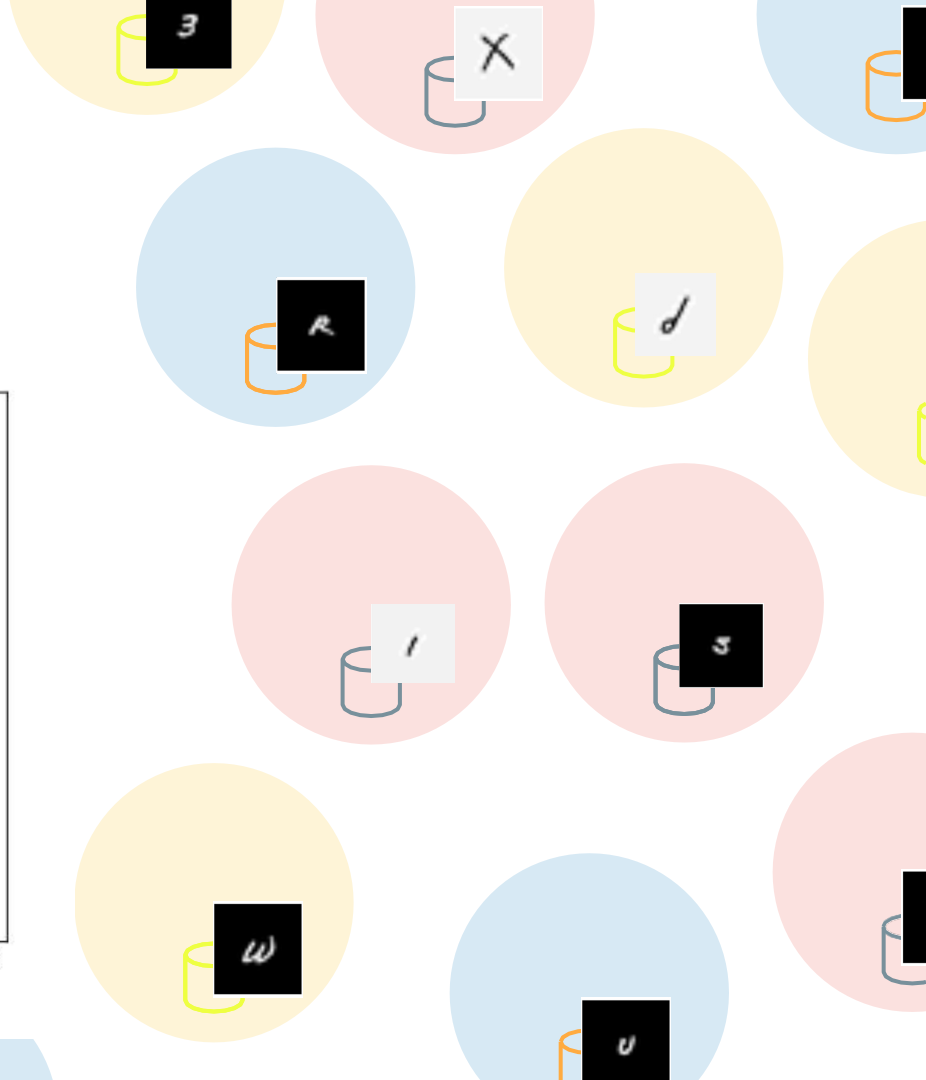
In **federated/private ML**:
•You **can't easily inspect** real user data.
•Debugging becomes **very difficult** without violating privacy.

Augenstein, et. al. **Generative Models for Effective ML on Private, Decentralized Datasets.** Arxiv, 2019.

Example bug

After application update, the classification accuracy drops

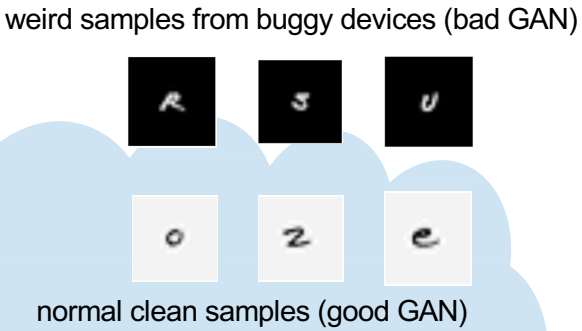


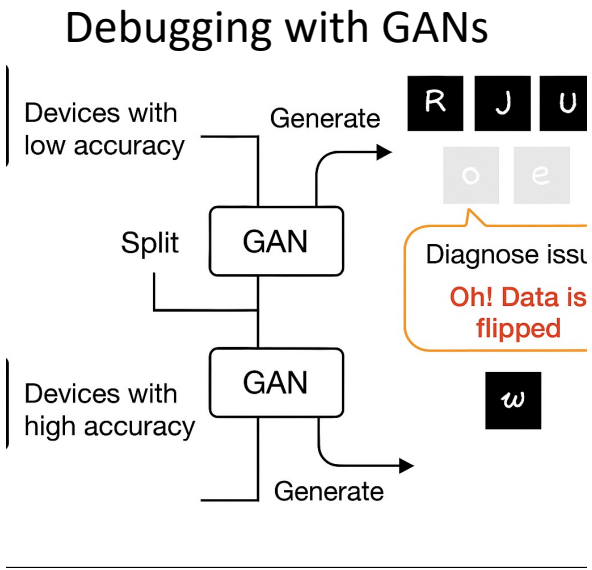


Example solution

Generative Adversarial Network

A type of machine learning framework, specifically a generative model, used to create new data instances that resemble the training data.
Used in image generation, text to image synthesis, etc.





- **Split** the device updates into two groups:
- Devices with **high classification accuracy**
- Devices with **low classification accuracy**
- **Train a GAN** (Generative Adversarial Network) on each group separately:
  - One GAN learns to **generate "good" data** (normal users).
  - One GAN learns to **generate "bad" data** (buggy or corrupted users).

Results

bad group

Population Description

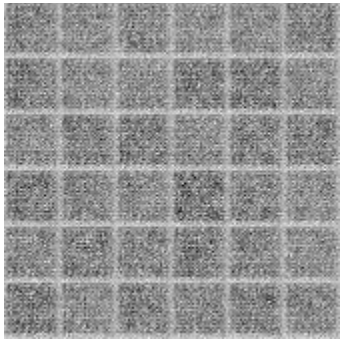
Sub-Population Description

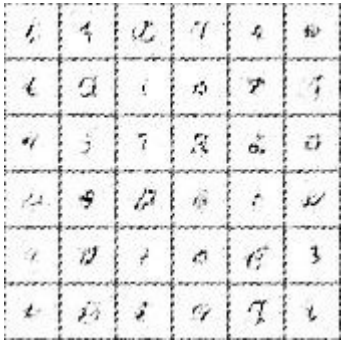
Example of Real Data on Devices in Sub-Population

After 1 round

After 1000 rounds

Devices where data classifies with ‘low’ accuracy

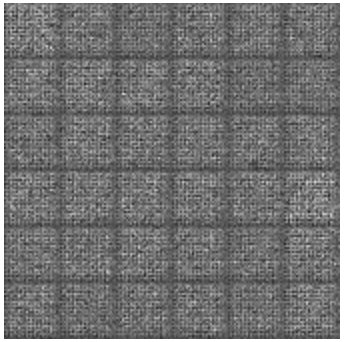




EMNIST Dataset, 50% of Devices have their images ‘flipped’ (black<-> white)

good group

Devices where data classifies with ‘high’ accuracy



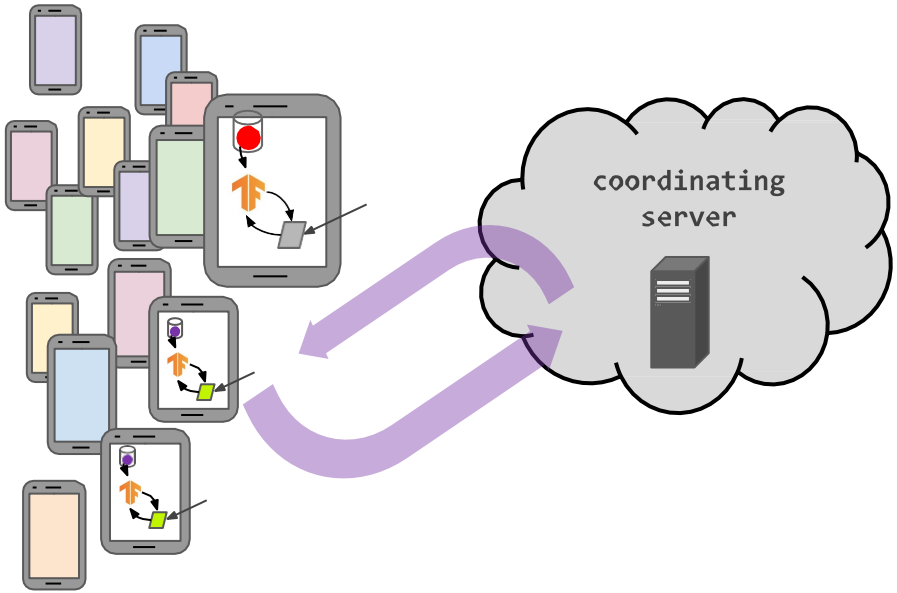
Augenstein, et. al. **Generative Models for Effective ML on Private, Decentralized Datasets.** Arxiv, 2019.

What do we do with these results?

Step	Action
Train GANs	Separately on **low-accuracy** and **high-accuracy** client groups.
Generate synthetic samples	After enough rounds (e.g., 1000), **visualize** the samples from each GAN.
Inspect the outputs	Look for **patterns**: Are they flipped? Are they weird? Are they missing classes?
Diagnose the issue	Example: "Oh! Devices with low accuracy have inverted images!"
Fix the problem	Update the application, training setup, or data pipeline to fix the root cause.

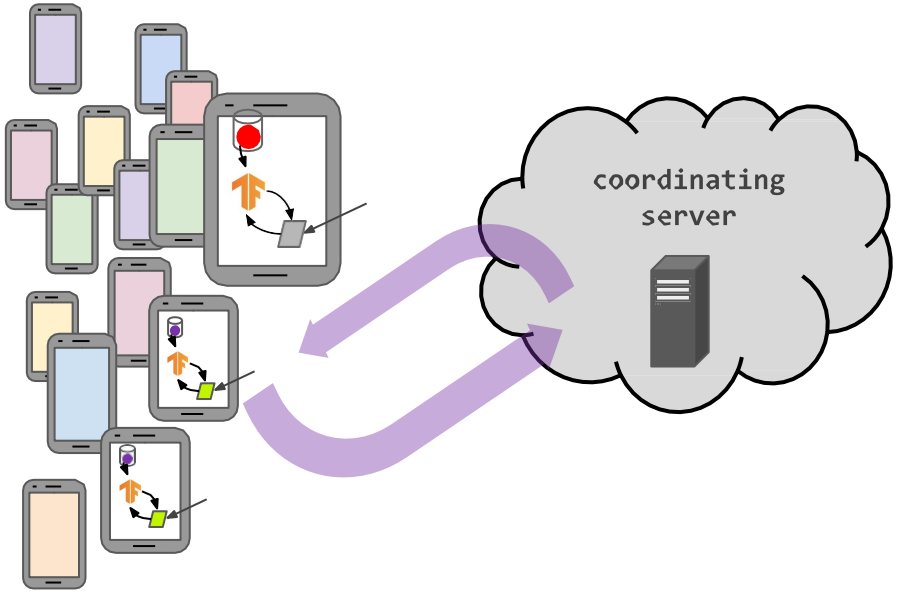
Data Poisoning

Adversary corrupts training data that is used during on-device training



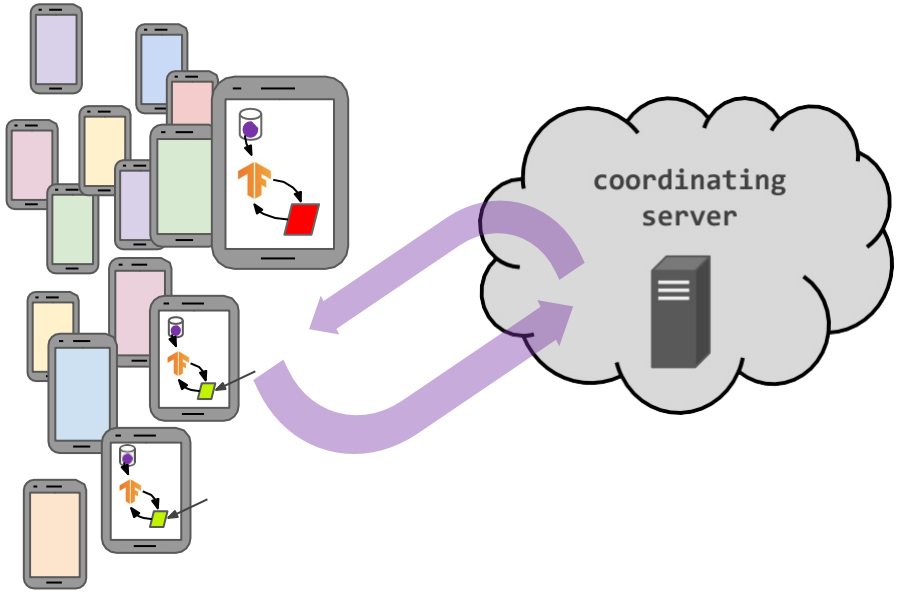
Data Poisoning

Adversary corrupts training data that is used during on-device training



Model Poisoning

Adversary compromises training binaries and corrupts the model being trained on-device



Data Poisoning

- Adversary injects poisoned data into compromised clients
- Compromising clients is less expensive
- Impact of malicious gradients is limited due to domain constraint on data

Model Poisoning

- Adversary has to break into clients' training binaries
- Compromising clients is more expensive
- Unconstrained malicious gradients can be destructive

Targeted vs. untargeted

- Attacking objectives:
  - Byzantine (untargeted) attacks. break the model everywhere
    - Degrade the performance on the main task
    - Example: Random users sending bad updates just to make the model worse at everything.
  - Backdoor (targeted) attacks, cause errors on specific triggers while the model looks fine otherwise.
    - Insert a backdoor task while maintaining a good overall accuracy
    - Example:
    - Model is perfect normally.
    - But if it sees a car with a racing stripe (left), it wrongly classifies it. Or cars painted green (middle) trick it. Or vertical stripes on background walls (right) fool the model.

i) cars with racing stripe

ii) cars painted in green

iii) vertical stripes on background wall

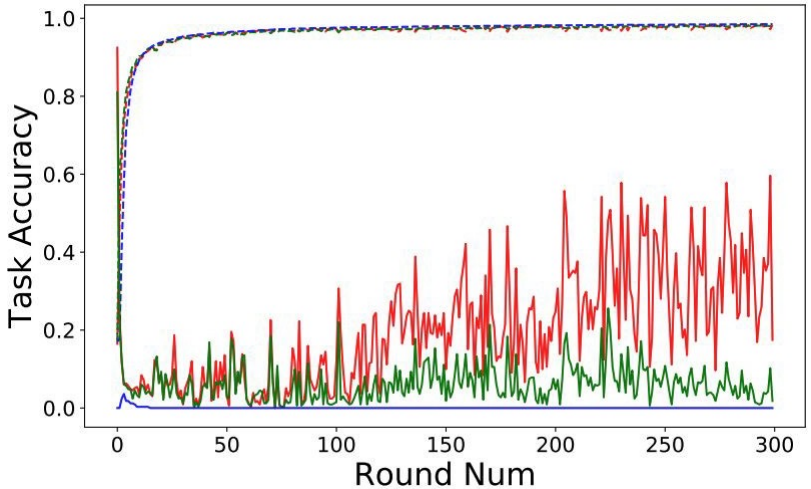
Bagdasaryan, et. al. **How to Backdoor Federated Learning.** AISTATS'20.

Can you backdoor federated learning?

- Attacking objectives:
  - Byzantine (untargeted) attacks:
    - Degrade the performance on the main task
  - **Backdoor (targeted) attacks**
    - Insert a backdoor task while maintaining a good overall accuracy
- Attacking strategies:
  - Data poisoning:
    - Insert poisoned data points
  - **Model poisoning:**
    - Send arbitrarily poisoned gradients
- Paper available on arXiv.
- Code available on GitHub.

Norm bounding and training with noise helps!

- EMNIST dataset
  - Handwritten digit classification, 3300+ users, each has a non-iid dataset of size 100+
- Backdoor task: images of ‘7’s from 20 random users labeled as ‘1’
  - Benign clients also have variants of these images with clean labels
- Model poisoning attacks via norm bounded model replacement
- Number of clients per round = 30
- **1 attacker in every round**

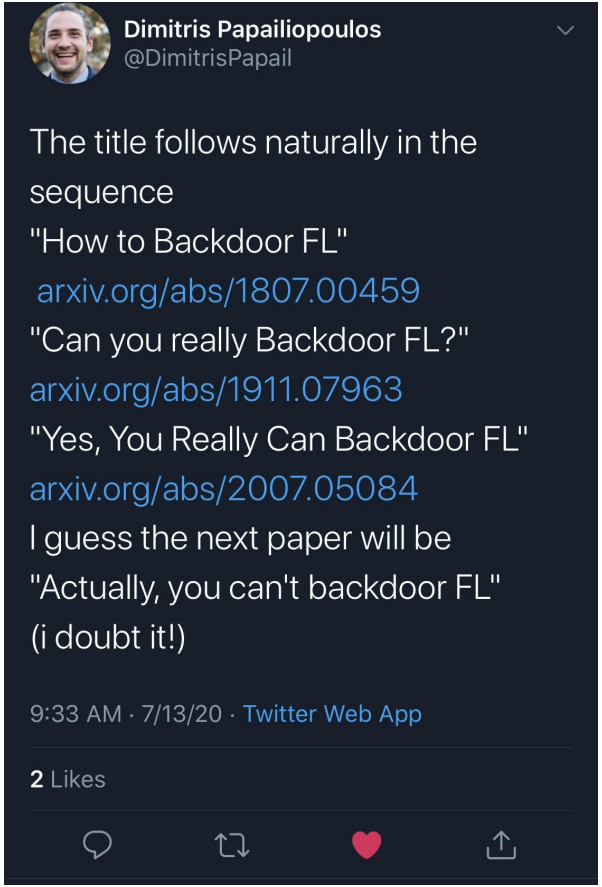


- main task: dashed lines
- backdoor task: dotted lines
- unattacked tasks
- norm bound = 5, no noise
- norm bound = 5, noise std = 0.025

The success of backdoor tasks dropped (From red to green)

Open challenges

- Heterogeneity in terms of:
  - Trust models, attacker capability, attacker objectives, and attack algorithms. E.g. model vs data poisoning, nation state vs compromised device, cross-device vs cross-silo FL
  - Attacking the tails is easier than the head
- Call to action?
  - Need benchmark problems & realistic scenarios.
  - What robustness matters in practice? Consider poisoning scales that are plausible given the threat actors and existing defenses
- Challenges:
  - adaptive adversaries, defenses with certified robustness, and tradeoffs between privacy mechanisms and robustness



Modeling heterogeneous data: *fairness & personalization*

RECALL

Federated ERM objective

Risk

$$R(h) = \mathbb{E}_{k \sim Q} \mathbb{E}_{(x,y) \sim P_k} [\ell(h(x; w), y)]$$

We have a sample of data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ **generated from *m* devices**

Each device may generate data according to its own distribution, P_k

Goal is to estimate the true risk over this sample, i.e.:

Empirical Risk

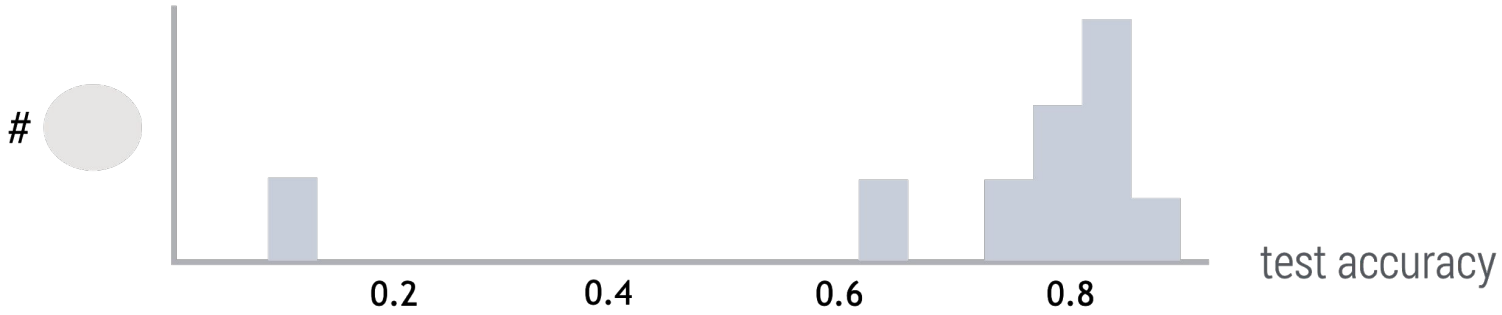
$$R_{emp}(h) = \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$$

FL: traditional empirical risk minimization

$$\textit{ERM:} \quad \min_{w} \quad \left(p_1 F_1 + p_2 F_2 + \cdots + p_m F_m \right)$$

potential issues:

- no accuracy guarantees for individual devices
- performance may vary widely across network



FL: traditional empirical risk minimization

$$\text{ERM:} \quad \min_{w} \left(p_1 F_1 + p_2 F_2 + \cdots + p_m F_m \right)$$

potential issues:

- no accuracy guarantees for individual devices
- performance may vary widely across network

Can we encourage a more fair (i.e., uniform) distribution of the model performance across devices?

Fair resource allocation objective

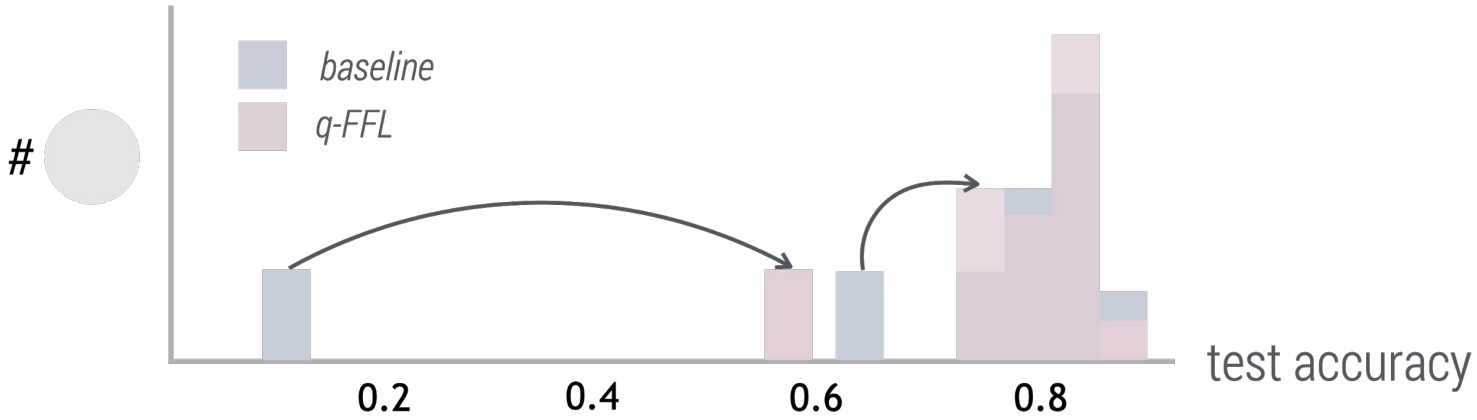
$$\textit{q-FFL:} \quad \min_{w} \frac{1}{q+1}\left(p_1 F_1^{q+1} + p_2 F_2^{q+1} + \cdots + p_m F_m^{q+1}\right)$$

- inspired by α-fairness for fair resource allocation in wireless networks
- a tunable framework ($q \to 0$: previous objective; $q \to \infty$: minimax fairness*)
- theory: increasing q results in more uniform accuracy distributions (e.g., reduced variance)

[Li et al, Fair Resource Allocation in Federated Learning, ICLR 2020]
*[Mohri, Sivek, Suresh, Agnostic Federated Learning, ICML 2019]
*[Hashimoto et al, Fairness without Demographics in Repeated Loss Minimization, ICML 2018]

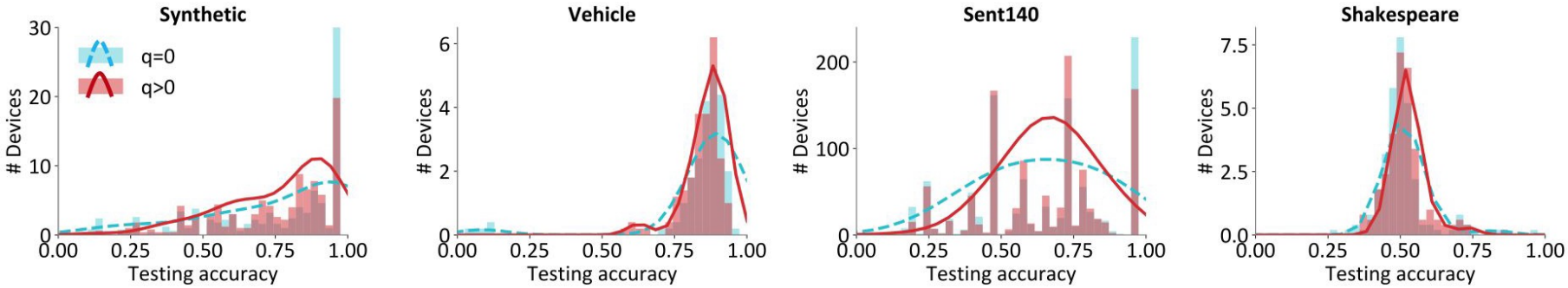
Fair resource allocation objective

$$\text{q-FFL:} \quad \min_{w} \frac{1}{q+1}\left(p_1 F_1^{q+1} + p_2 F_2^{q+1} + \cdots + p_m F_m^{q+1}\right)$$



[Li et al, Fair Resource Allocation in Federated Learning, ICLR 2020]

Fair resource allocation objective



on average,

can cut variance in half

while maintaining mean accuracy

[Li et al, Fair Resource Allocation in Federated Learning, ICLR 2020]

[Caldas et al, LEAF: A Benchmark for Federated Settings, 2019, leaf.cmu.edu]

RECALL

Federated ERM objective

Risk

$$R(h) = \mathbb{E}_{k \sim Q} \mathbb{E}_{(x,y) \sim P_k} [\ell(h(x; w), y)]$$

We have a sample of data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ **generated from *m* devices**

Each device may generate data according to its own distribution, P_k

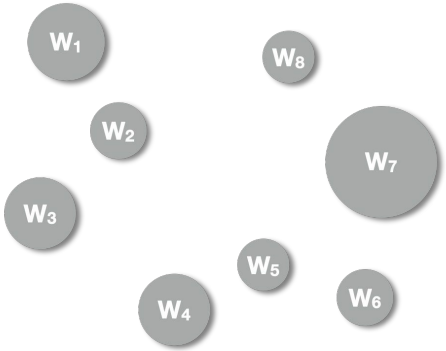
Goal is to estimate the true risk over this sample, i.e.:

Empirical Risk

$$R_{emp}(h) = \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$$

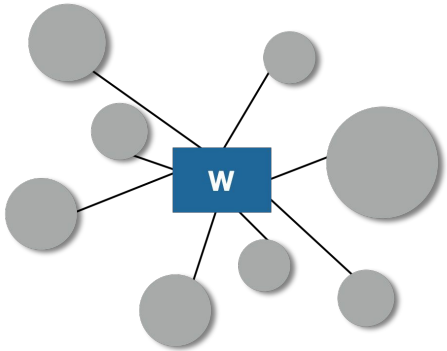
How to model federated data?

local



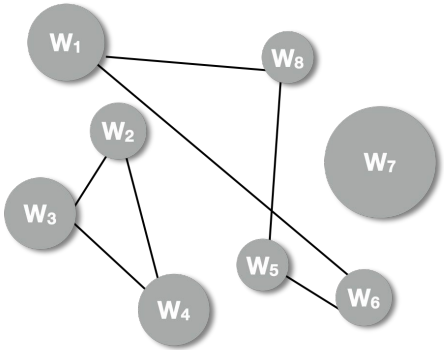
✓ *personalized models*
✗ *don't learn from peers*

global



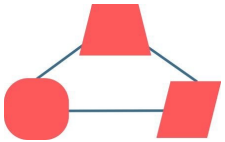
✗ *non-personalized models*
✓ *learn from peers*

??



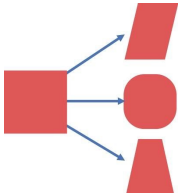
✓ *personalized models*
✓ *learn from peers*

Approaches for personalization



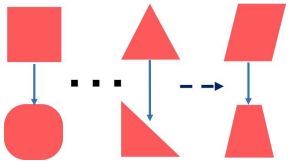
Multi-task learning

- Jointly learn shared, yet personalized models



Fine-tuning

- Learn a global model, then “fine-tune”/adapt it on local data
- See also: transfer learning, domain adaptation



Meta learning (initialization-based)

- Learn initialization over multiple tasks, then train locally

Federated multi-task learning

- Given heterogeneity of data, can view each device/client's training objective as a separate 'task'

- Multi-task learning (MTL):
  "*improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias*"*

[Smith et al, Federated Multi-Task Learning, NeurIPS 2017]
*[Caruana, R, Multitask learning, Machine Learning, 1997]
[Caruana, R, Multitask learning: A knowledge-based source of inductive bias, ICML, 1993]

Multi-task learning

$$\min_{\mathbf{W},\boldsymbol{\Omega}} \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t(\mathbf{w}_t, \mathbf{x}_t^i) + \mathcal{R}(\mathbf{W}, \boldsymbol{\Omega})$$

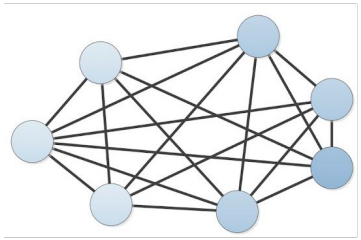
models

task relationship

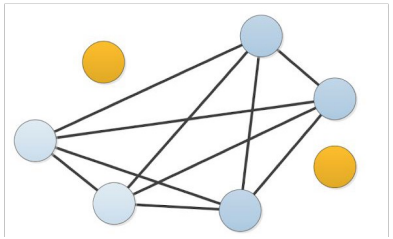
losses

regularizer

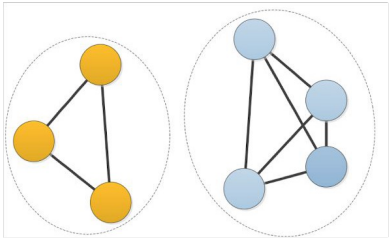
all tasks related



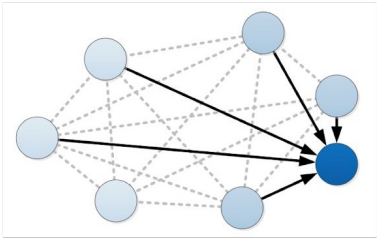
outlier tasks



clusters / groups

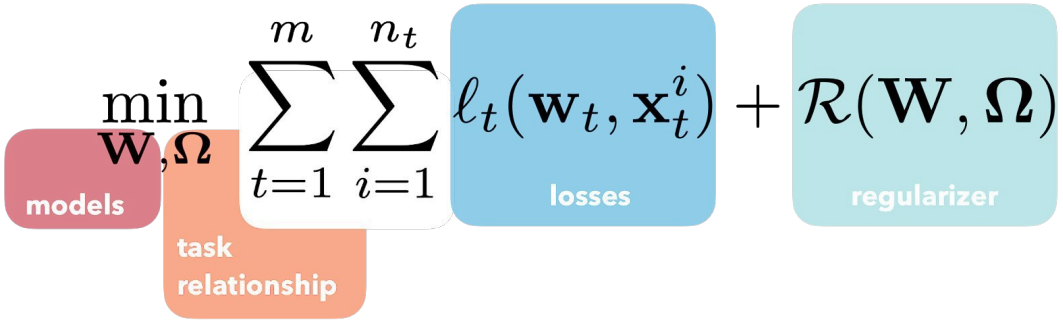


asymmetry



[MTL: Theory, Algorithms, and Applications, Zhou, Chen, Li, SDM Tutorial, 2012]

Multi-task learning



Relationship MTL, e.g., mean-regularized

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda_1 \ \mathrm{tr}\left(\mathbf{W}\mathbf{\Omega}\mathbf{W}^T\right) + \lambda_2 \|\mathbf{W}\|_F^2$$

Cluster-regularized MTL

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda \ \mathrm{tr}\left(\mathbf{W}(\eta\mathbf{I} + \mathbf{\Omega})^{-1}\mathbf{W}^T\right)$$

$$\mathbf{\Omega} \in \mathcal{Q} = \left\{\mathbf{Q} \mid \mathbf{Q} \succeq \mathbf{0}, \ \mathrm{tr}(\mathbf{Q}) = k, \ \mathbf{Q} \preceq \mathbf{I}\right\}$$

MOCHA: federated multi-task learning

$$\min_{\mathbf{W},\mathbf{\Omega}} \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t(\mathbf{w}_t^T \mathbf{x}_t^i) + \mathcal{R}(\mathbf{W}, \mathbf{\Omega})$$

- Solve for W, $\mathbf{\Omega}$ in an alternating fashion
- Modify CoCoA to solve W in the federated setting

dual

$$\min_{\boldsymbol{\alpha}} \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t^*(-\boldsymbol{\alpha}_t^i) + \mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha})$$

subproblem

$$\min_{\Delta\boldsymbol{\alpha}_t} \sum_{i=1}^{n_t} \ell_t^*(-\boldsymbol{\alpha}_t^i - \Delta\boldsymbol{\alpha}_t^i) + \langle \mathbf{w}_t(\boldsymbol{\alpha}), \mathbf{X}_t \Delta\boldsymbol{\alpha}_t \rangle + \frac{\sigma'}{2} \|\mathbf{X}_t \Delta\boldsymbol{\alpha}_t\|_{\mathbf{M}_t}^2$$

[Smith et al, Federated Multi-Task Learning, NeurIPS 2017]

Meta learning & federated learning

Algorithm 1 Connects FL and MAML (left), Reptile Batch Version(middle), and FedAvg (right).

OuterLoop/Server learning rate α
InnerLoop/Client learning rate β
Initial model parameters θ
while not done **do**
  Sample batch of tasks/clients $\{T_i\}$
  for Sampled task/client T_i **do**
    if FL **then**
      $g_i, w_i = ClientUpdate(\theta, T_i, \beta)$
    else if MAML **then**
      $g_i = InnerLoop(\theta, T_i, \beta)$
    end if
  end for
  if FL **then**
    $\theta = ServerUpdate(\theta, \{g_i, w_i\}, \alpha)$
  else if MAML **then**
    $\theta = OuterLoop(\theta, \{g_i\}, \alpha)$
  end if
end while

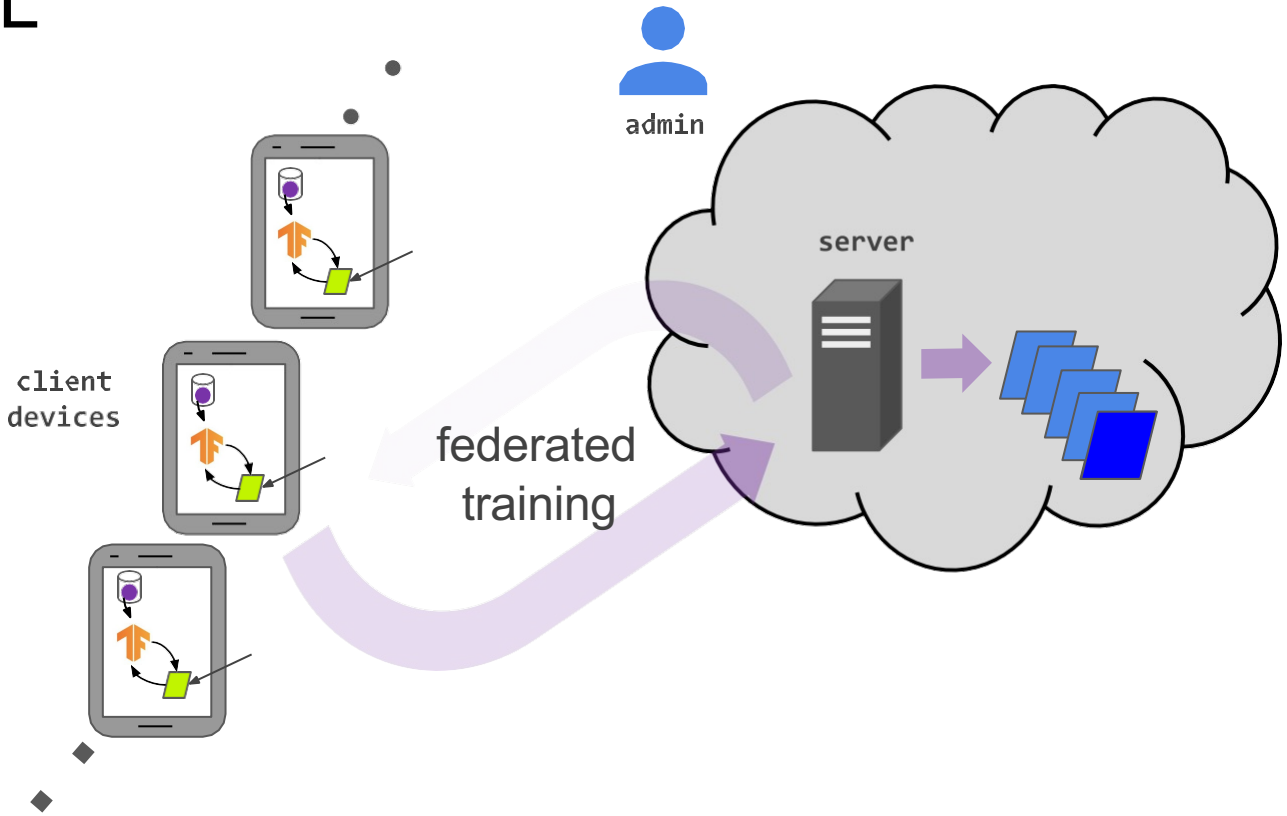
Require: : Reptile Step K.
  function $InnerLoop(\theta, T_i, \beta)$
    Sample K-shot data $D_{i,k}$ from T_i.
    $\theta_i = \theta$
    for each local step i from 1 to K **do**
      $\theta_i = \theta_i - \beta \nabla_\theta L(\theta_i, D_{i,k})$
    end for
    Return $g_i = \theta_i - \theta$
  end function
Require: : Meta Batch Size M.
  function $OuterLoop(\theta, \{g_i\}, \alpha)$
    $\theta = \theta + \alpha \frac{1}{M} \sum_{i=1}^{M} g_i$
    Return θ
  end function

Require: FedAvg Local Epoch E.
  function $ClientUpdate(\theta, T_i, \beta)$
    Split local dataset into batches B
    $\theta_i = \theta$
    for each local epoch i from 1 to E **do**
      for batch $b \in B$ **do**
        $\theta_i = \theta_i - \beta \nabla_\theta L(\theta_i, b)$
      end for
    end for
    Return $g_i = \theta_i - \theta$
  end function
Require: Clients per training round M.
  function $ServerUpdate(\theta, \{g_i, w_i\}, \alpha)$
    $\theta = \theta + \alpha \sum_{i=1}^{M} w_i g_i / \sum_{i=1}^{M} w_i$
    Return θ
  end function

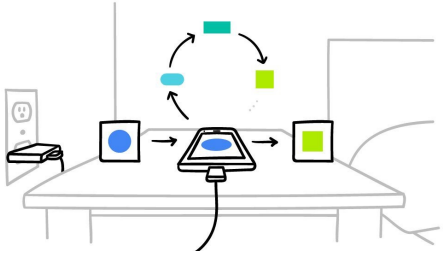
[Jiang et al, Improving federated learning personalization via model agnostic meta learning, 2019]
[Khodak, Balcan, Talwalkar, Adaptive gradient-based meta-learning methods, NeurIPS 2019]

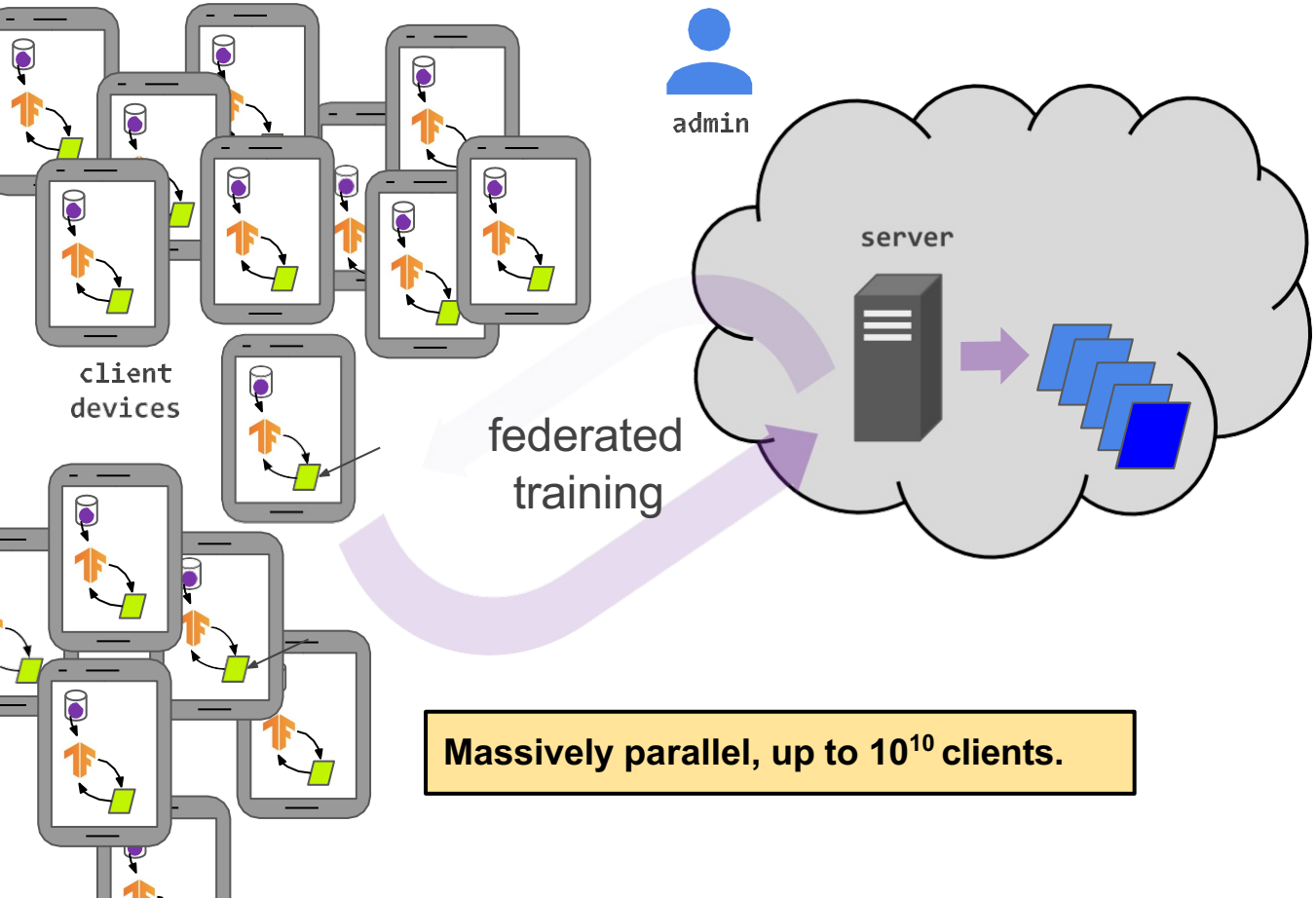
System Challenges

System challenges in cross-device FL



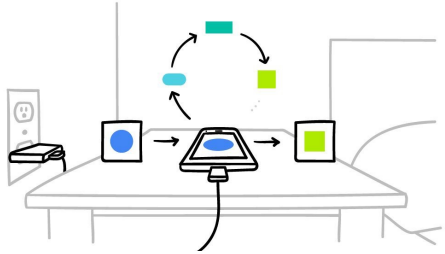
Cross-device federated learning

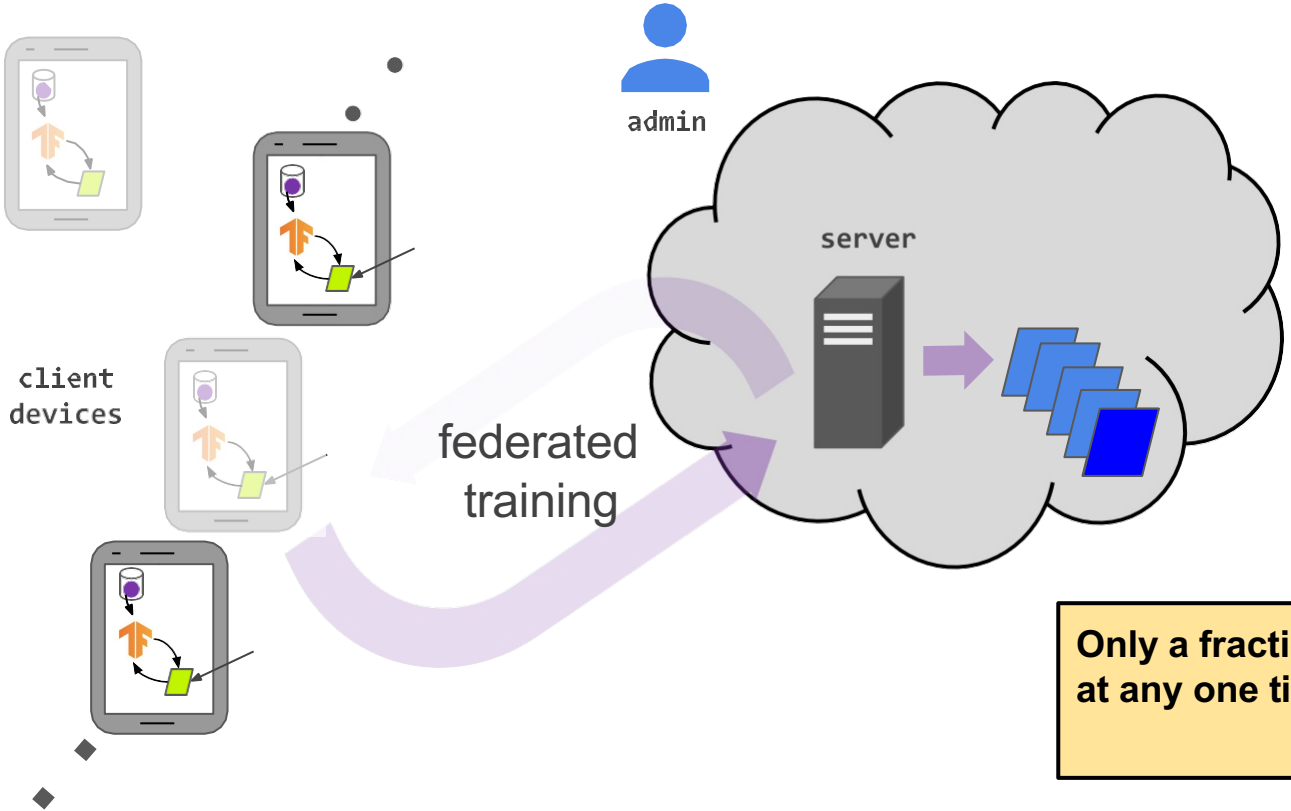




Massively parallel, up to 10^{10} clients.

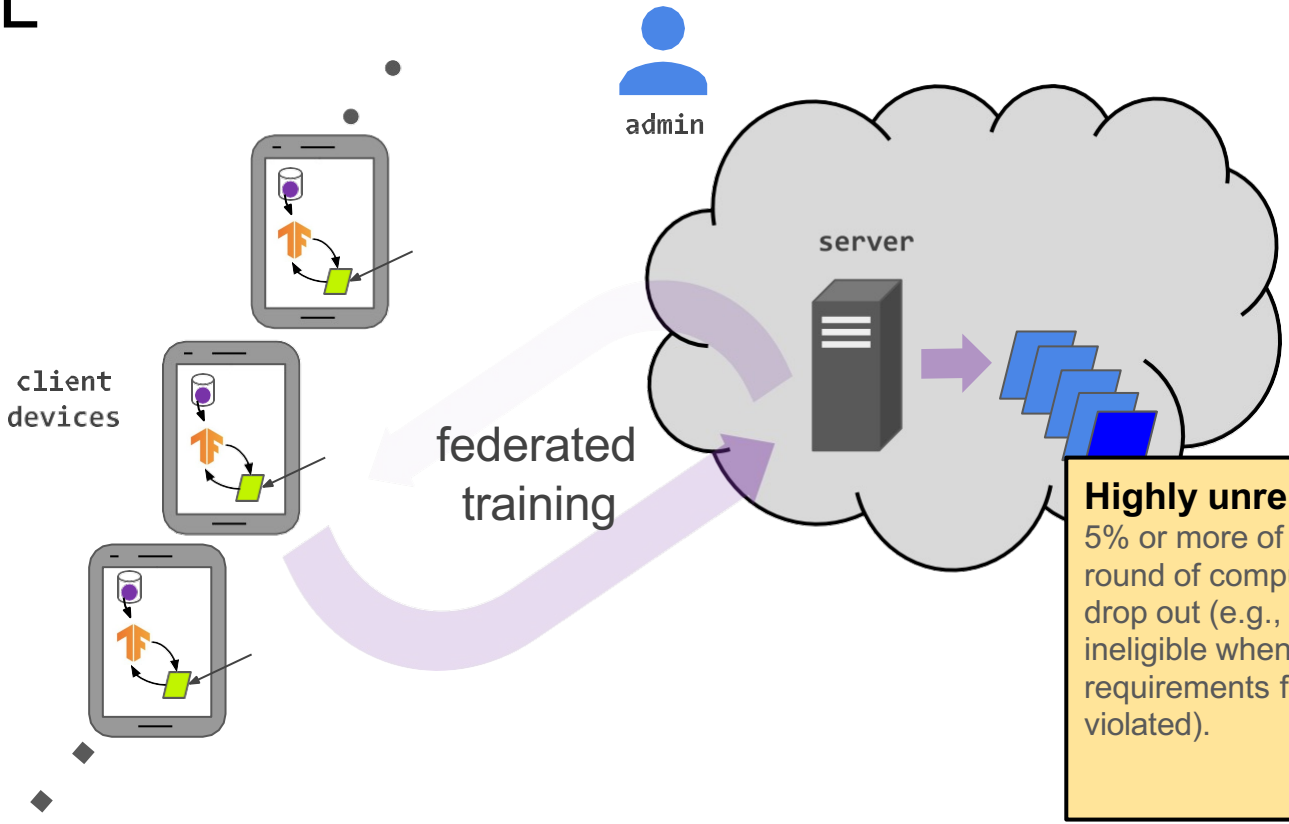
Cross-device federated learning





Only a fraction of clients are available at any one time.

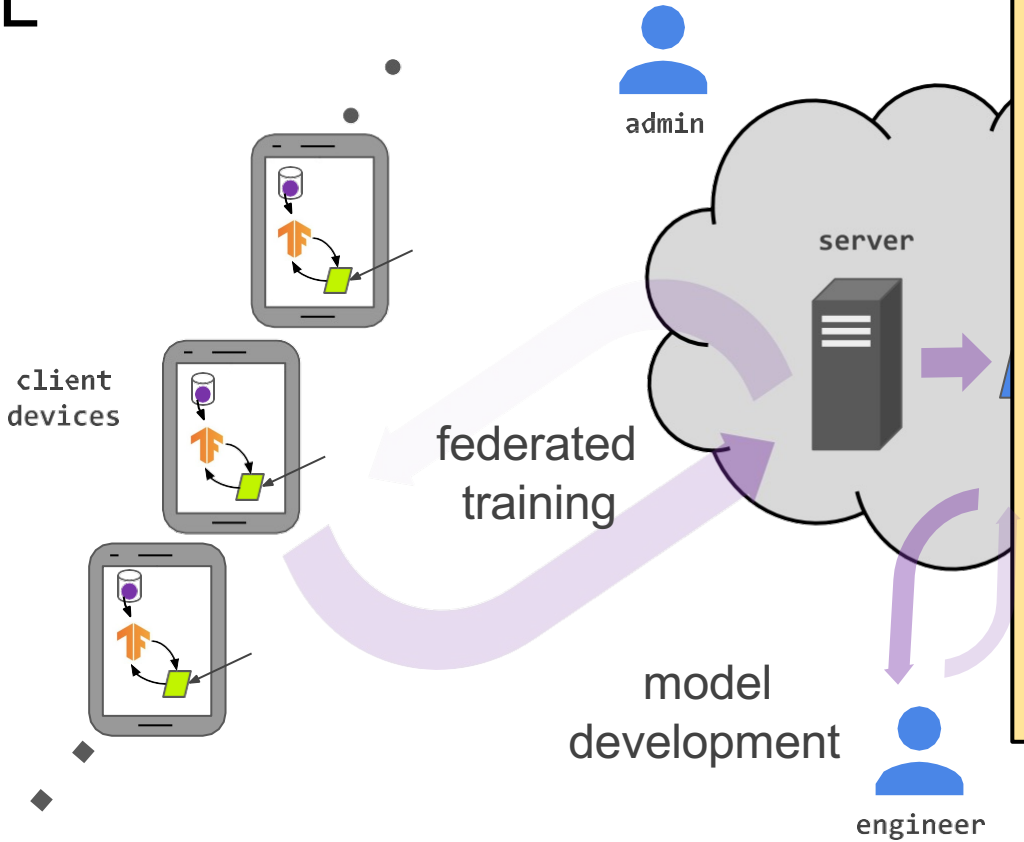
System challenges in cross-device FL



Highly unreliable

5% or more of the clients participating in a round of computation are expected to fail or drop out (e.g., because the device becomes ineligible when battery, network, or idleness requirements for training/computation are violated).

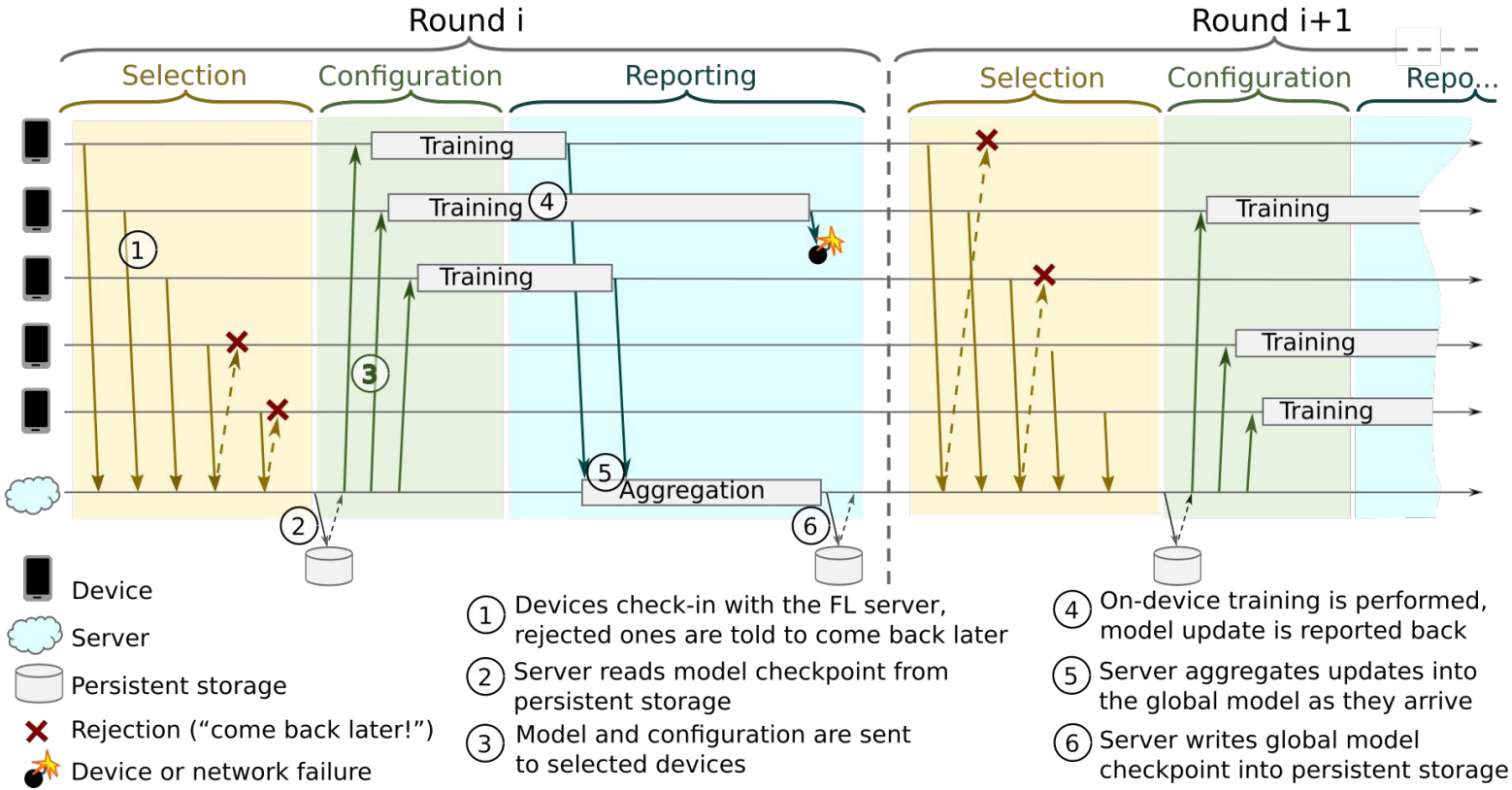
System challenges in cross-device FL



Operational Challenges

- Ensuring compatibility of deployed code (e.g. TensorFlow graphs) with multiple versions of the deployed runtime.
- Ensuring stability of user devices in all circumstances: no impact on performance, battery, or network usage.
- Providing a multi-tenant service used by multiple teams, each training many models.
- Supporting enabling workflows: analytics, model debugging, hyperparameter search, etc.

An example cross-device federated learning protocol



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

tff.federated_computation

Specification of a distributed computation, not code that runs in any one place.

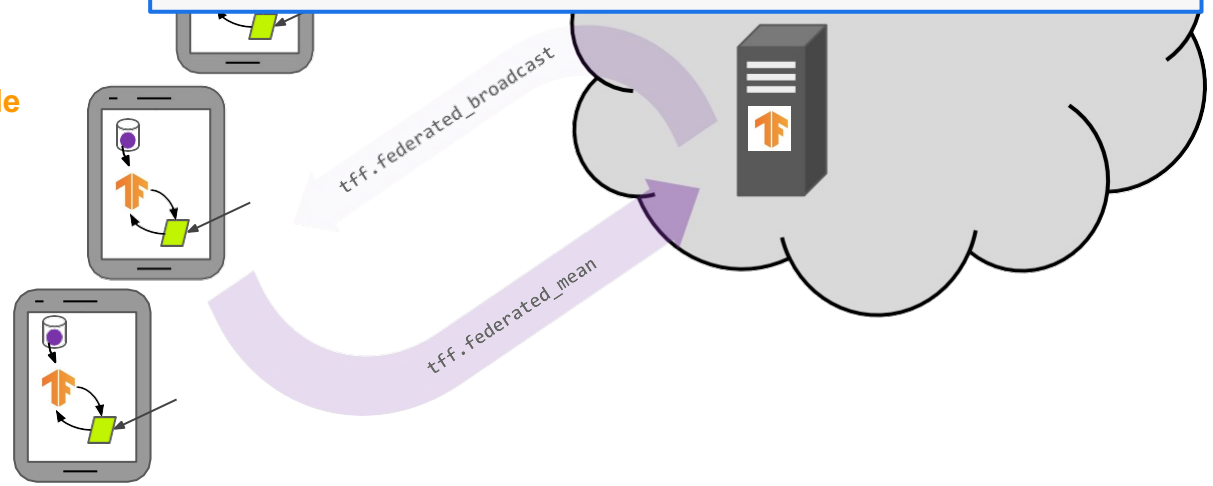
No enumeration or indexing of clients.

Values have types indicating @SERVER, @CLIENTS.

Think of the values as *handles* or *references* to data that may live in many places.

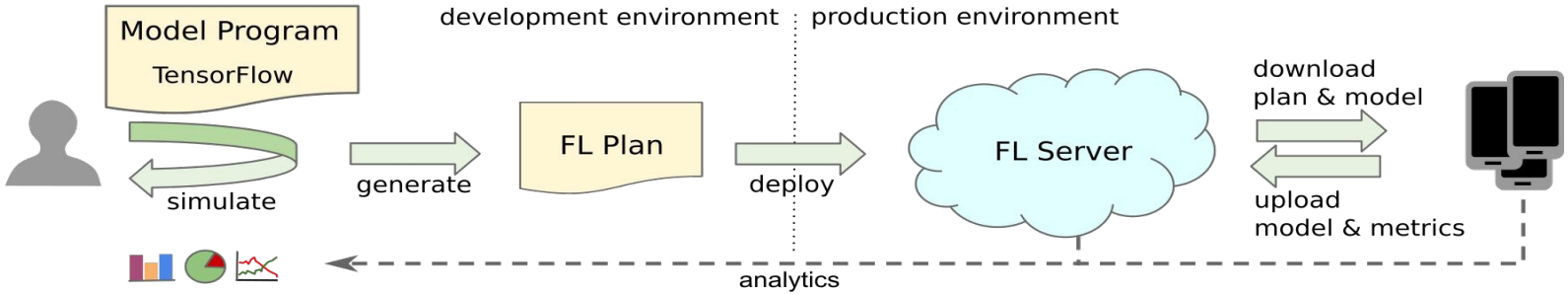
Declarative aggregation and broadcast operators amenable to large-scale multi-tier distributed architectures.

```
@tff.federated_computation(federated_server_type, federated_dataset_type)
def next_fn(server_weights, federated_dataset):
  server_weights_at_client = tff.federated_broadcast(server_weights)

  client_weights = tff.federated_map(
      client_update_fn, (federated_dataset, server_weights_at_client))

  mean_client_weights = tff.federated_mean(client_weights)

  server_weights = tff.federated_map(server_update_fn, mean_client_weights)

  return server_weights
```



https://www.tensorflow.org/federated/tutorials/building_your_own_federated_learning_algorithm

Developer workflows in federated learning

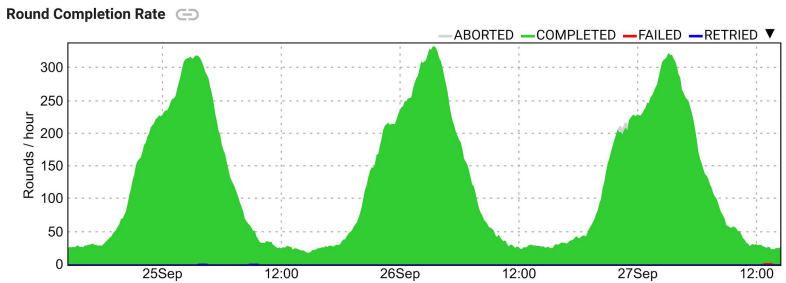


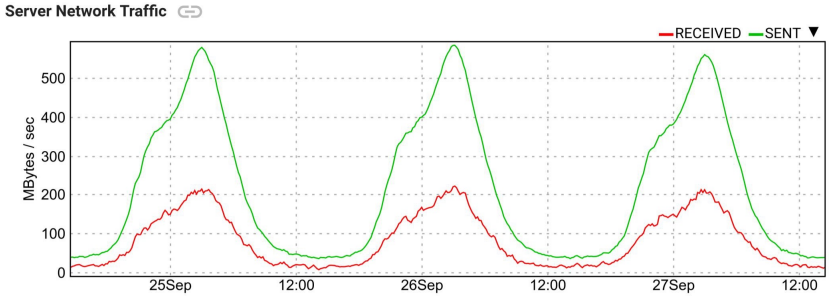
- Model developers depend on the production system for experimentation
  - They only have access to proxy data but not to the real data
  - Develop in Python, then push the result automatically to production and get metrics back
- Experimentation must never affect the user experience on devices
  - Training has no visible effect to the user -- inference models are manually pushed
  - Device architecture ensures that device health is not affected

Population scales and characteristics

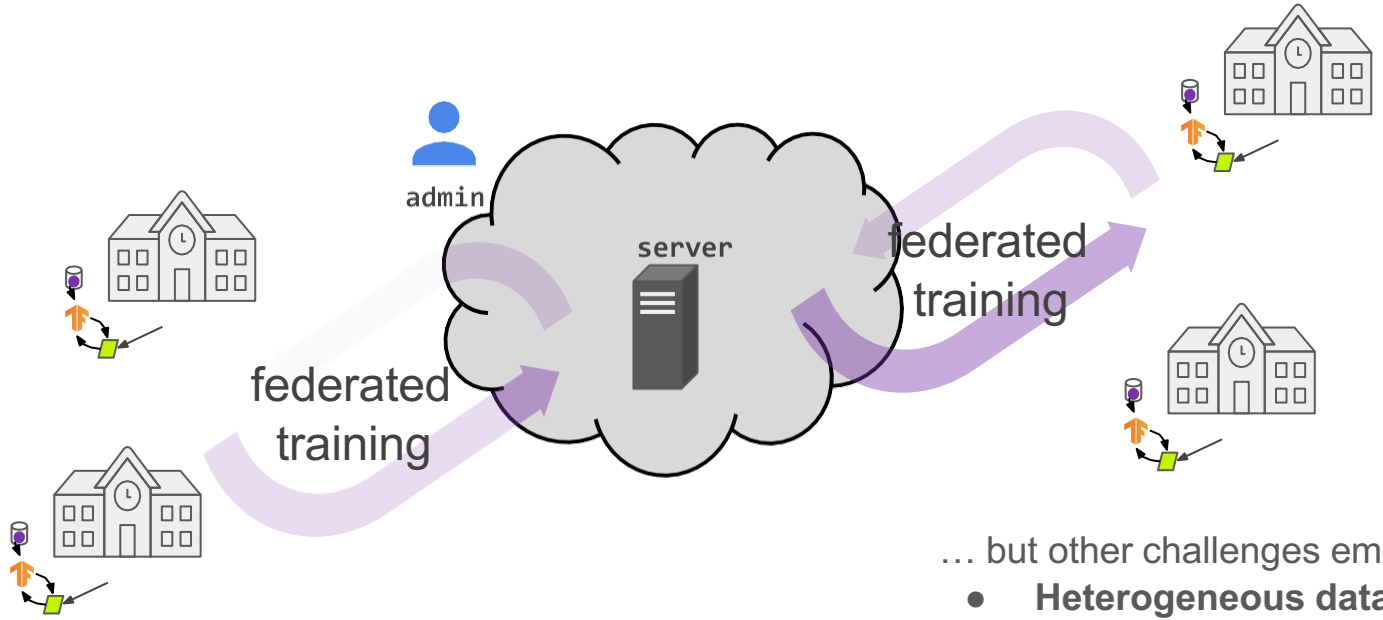
Problem: Population size varies drastically

- Diurnal patterns for regions
- Different lifecycle stages and types of applications
  - Development: <= 12 devices
  - Dogfooding: <= 1000 devices
  - Production: > 10,000,000 devices (but largely varies by application)





System challenges in cross-silo federated learning



Many things are easier ...

- High reliability
- Most clients can participate in all rounds.
- Faster compute & networks

… but other challenges emerge

- **Heterogeneous data schemas** - different features, different labels, different formats
- Joins for vertical (feature) partitioned data
- Software deployment challenges (more complex than each client is running the same app)

Thanks!

Peter Kairouz

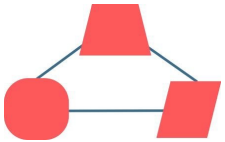
Brendan McMahan

Virginia Smith

https://sites.google.com/view/fl-tutorial

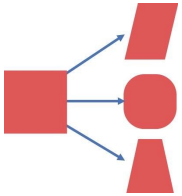
APPENDIX

Approaches for personalization



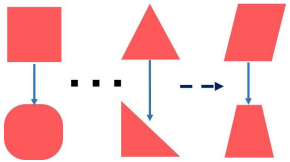
Multi-task learning

- Jointly learn shared, yet personalized models



Fine-tuning

- Learn a global model, then “fine-tune”/adapt it on local data
- See also: transfer learning, domain adaptation

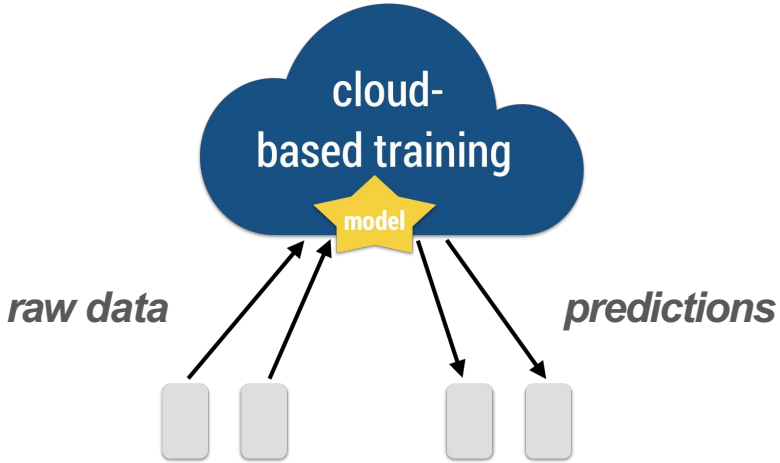


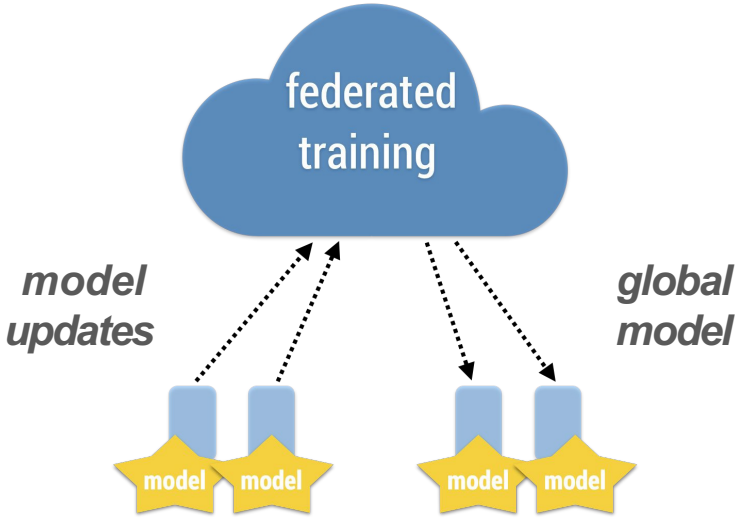
Meta learning (initialization-based)

- Learn initialization over multiple tasks, then train locally

Federated optimization

goal: train machine learning models at the edge

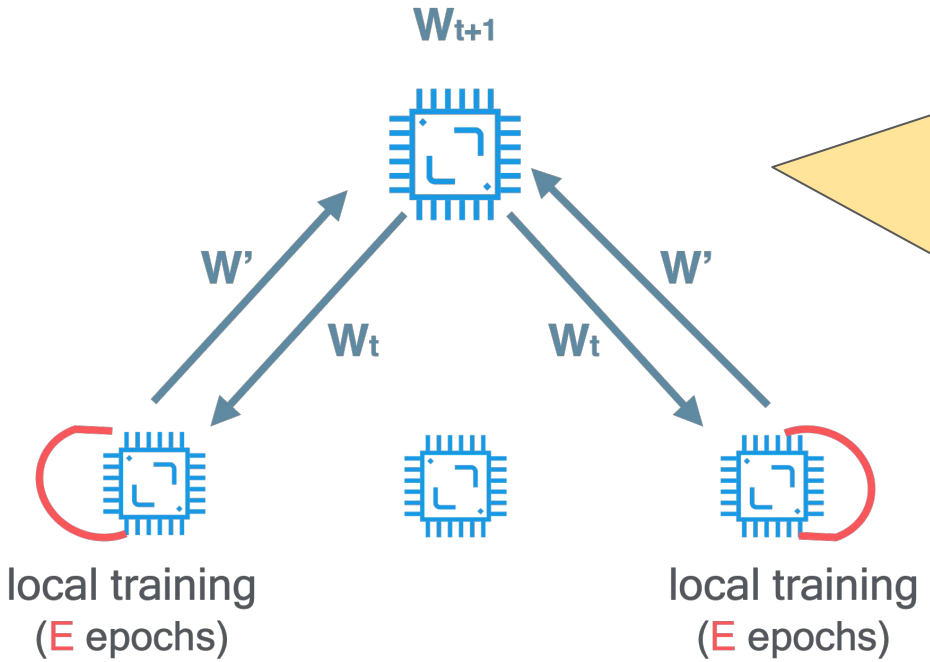




why? ✓ reduce strain on network ✓ privacy ✓ quickly incorporate new data

A STANDARD BASELINE

Federated Averaging (FedAvg)



- At each communication round:
  (i) run SGD locally, then
  (ii) average the model updates
- Can add privacy mechanisms to this procedure (more later …)
- Reduces communication by:
  (i) performing local updating,
  (ii) communicating with a subset of devices

Terminology

m	number of devices
N	total number of data points
n_k	number of data points on device k
$(x_k^{(i)}, y_k^{(i)})$	i^{th} data point on device k
w	model parameters

Empirical Risk

$$R_{emp}(h) = \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$$

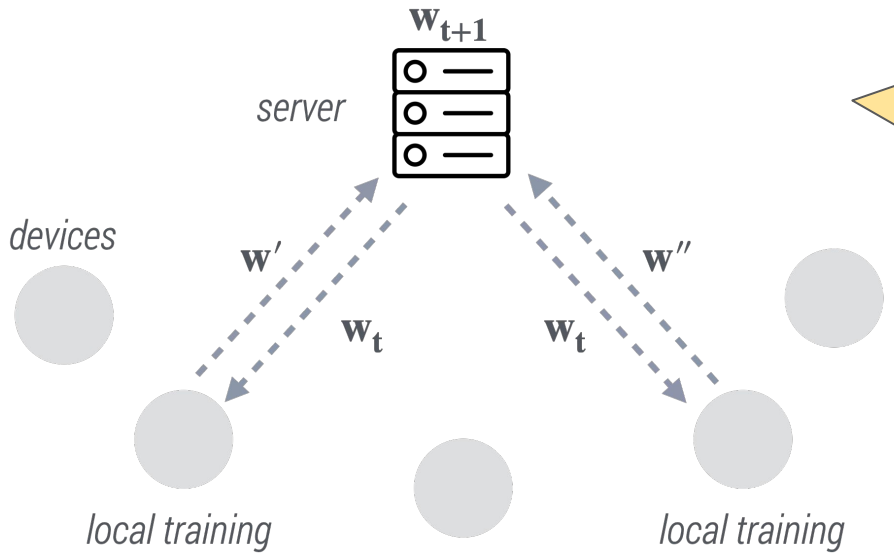
$$= \sum_{k=1}^{m} p_k F_k(w)$$

Empirical Risk

$$R_{emp}(h) = \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)}) = \sum_{k=1}^{m} p_k F_k(w)$$

A STANDARD BASELINE

Federated Averaging (FedAvg)



- At each communication round:
  (i) run SGD locally, then
  (ii) average the model updates
- Can add privacy mechanisms to procedure (more later …)
- Reduces communication by:
  (i) performing local updating,
  (ii) communicating with a subset of devices