CS 350S: Privacy-Preserving Systems

Federated learning

Recap: AOL query deanonymization attack

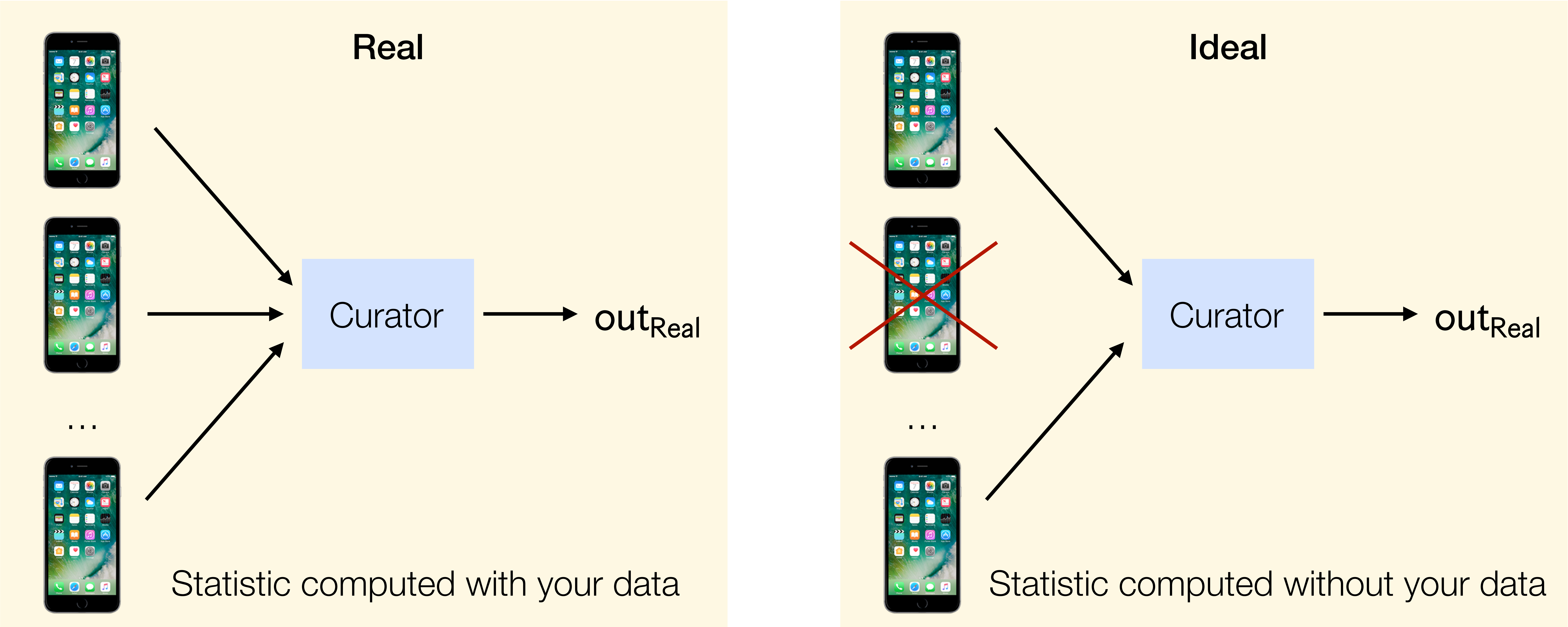
- AOL query dataset had >20M anonymized search queries from 650,000 AOL users over 3 months
- Dataset released where each username was replaced with a random identifier
- Queries for
  - “Landscapers in Lilburn, Ga”
  - Several people with last name Arnold
  - “Homes sold in shadow lake subdivision Gwinnett county Georgia”
  - … other sensitive queries
- Only 14 citizens with last name Arnold in Gwinnett County
- Found that user was Thelma Arnold, 62-year old woman in Georgia

The New York Times

A Face Is Exposed for AOL Searcher No. 4417749

https://www.nytimes.com/2006/08/09/technology/a-face-is-exposed-for-aol-searcher-no-4417749.html

Recap: Differential privacy



$\mathsf{out}_{\mathsf{Real}} \approx \mathsf{out}_{\mathsf{Ideal}}$ (Not cryptographic indistinguishability)

Recap: Differential privacy

[Dwork, Sherry, Nissim, Smith]

Mechanism $\mathscr{M} : \mathscr{X}^n \to \mathscr{Y}$
For database with n rows of type $\mathscr{X}$ and output statistic $\mathscr{Y}$

Two databases are "neighboring" if they differ in at most one row

A mechanism $\mathscr{M}$ is ε differentially private if for all pairs of "neighboring databases" D, D' and every set of values $S \in \mathscr{Y}$:

$$\text{Pr}[\mathscr{M}(D) \in S] \leq e^{\varepsilon} \cdot \text{Pr}[\mathscr{M}(D') \in S]$$

Outline

1. Federated learning
2. Flamingo
3. Other research topics
4. Student presentation

Goal: privately train a model across many users

One approach: generic multi-party computation

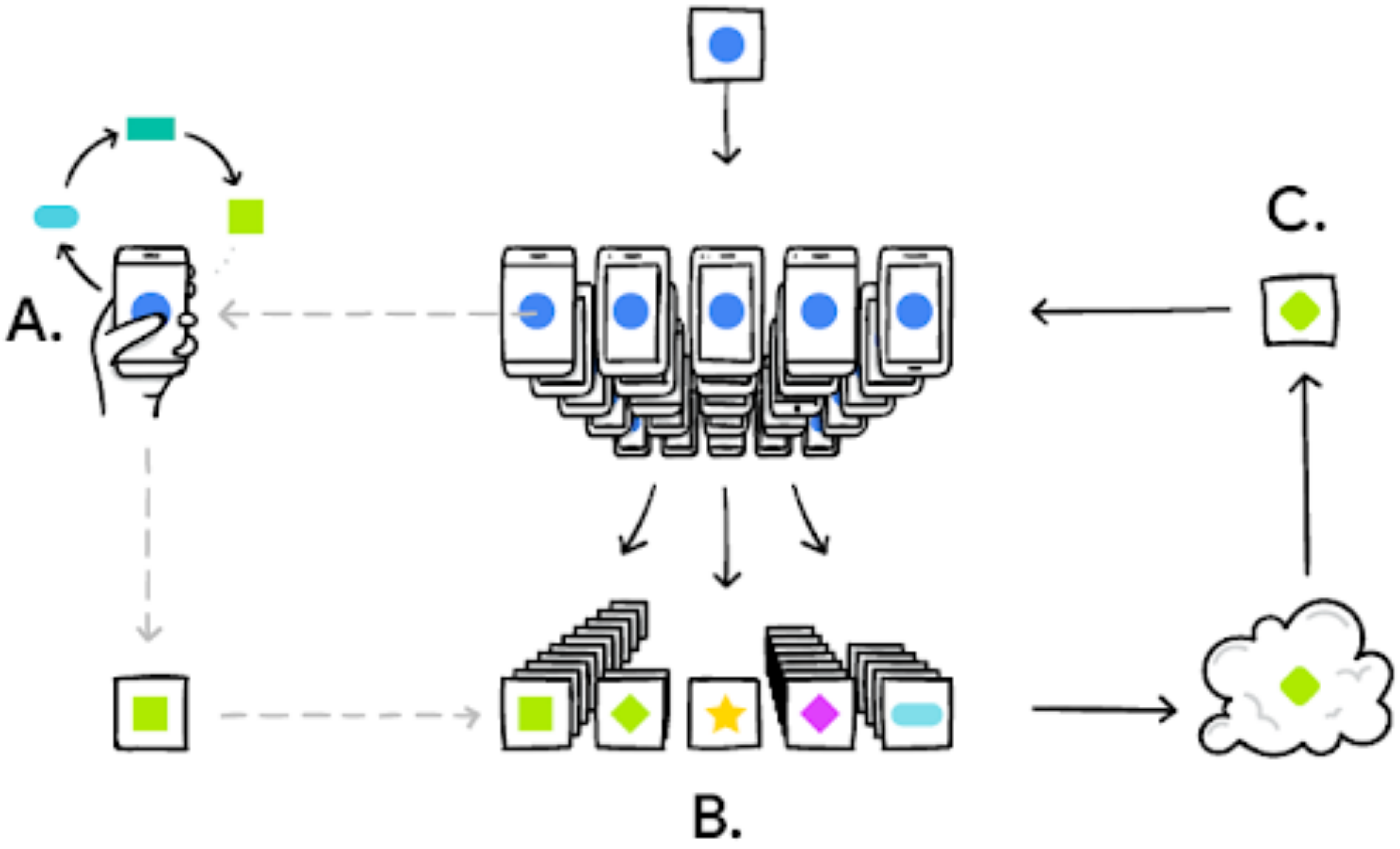
MPC can make it possible to compute arbitrary functions over arbitrary numbers of users

Why not run a MPC across all users?

- Clients are resource-constrained and might go offline
- Many MPC protocols don't scale well to many parties

Federated learning

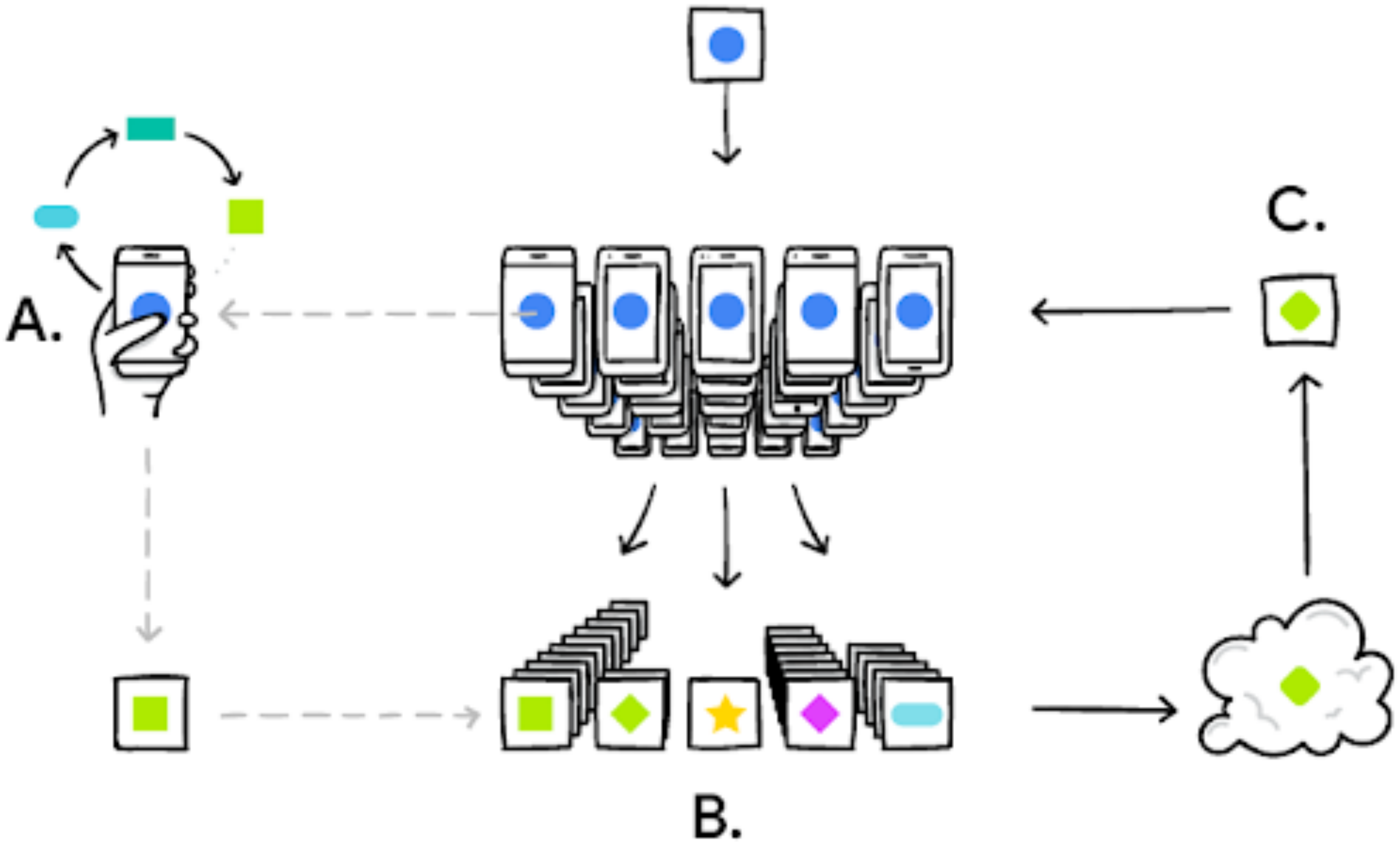
- Training data remains on the device
- Clients send updates to the server
- The server aggregates many user updates to produce a model update
- The server pushes the model update to the clients
- Repeat



https://research.google/blog/federated-learning-collaborative-machine-learning-without-centralized-training-data/

Challenge: resource-constrained, heterogeneous clients

- Mobile devices have powerful processors, but limited and intermittent connectivity (in contrast to datacenters, where compute is dominant factor)
- Data is not IID (user's dataset is not representative of population) and unbalanced (some users use the app more than others)
- Naively distributing a training algorithm like SGD does not satisfy these constraints



https://research.google/blog/federated-learning-collaborative-machine-learning-without-centralized-training-data/

Idea: more compute for higher-quality updates

[McMahan, Moore, Ramage, Hampson, Arcas]

- For each round, choose a set of clients
- Each chosen client runs some number of SGD updates locally
- Each chosen client sends its update to the aggregator

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
  initialize w_0
  for each round $t = 1, 2, \dots$ **do**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (random set of m clients)
    for each client $k \in S_t$ **in parallel do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    $m_t \leftarrow \sum_{k \in S_t} n_k$
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ // *Erratum*[4]

ClientUpdate(k, w): // *Run on client k*
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
  for each local epoch i from 1 to E **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(w; b)$
  return w to server

Idea: more compute for higher-quality updates

[McMahan, Moore, Ramage, Hampson, Arcas]

- For each round, choose a set of clients
- Each chosen client runs some number of SGD updates locally
- Each chosen client sends its update to the aggregator

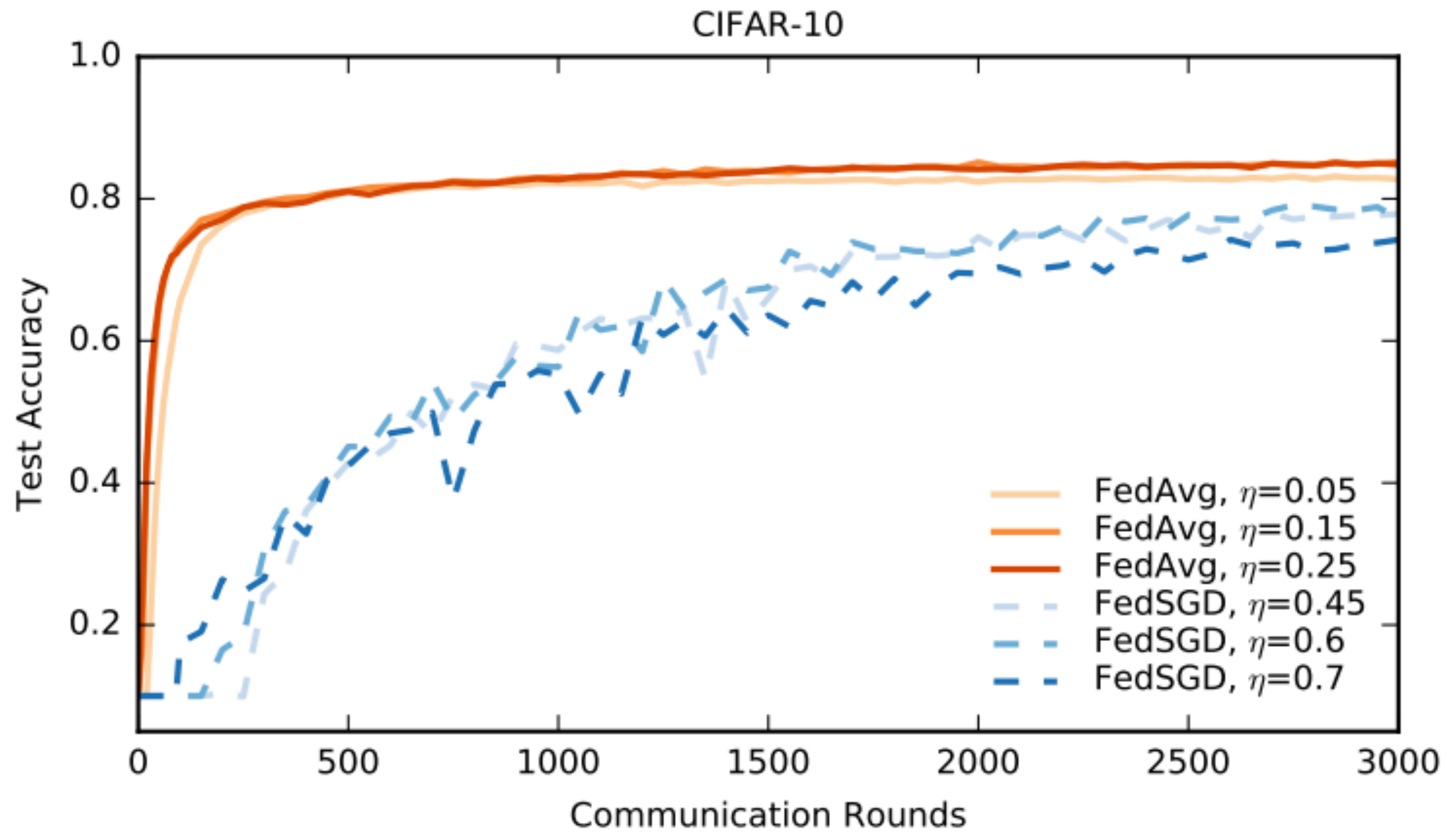


Figure 4: Test accuracy versus communication for the CIFAR10 experiments. `FedSGD` uses a learning-rate decay of 0.9934 per round; `FedAvg` uses $B = 50$, learning-rate decay of 0.99 per round, and $E = 5$.

Use-cases of federated learning at Google

- Gboard: Next-word prediction, emoji usage, out-of-vocabulary word discovery
- “Hey Google” detection models in Assistant
- Suggesting replies in Google messages
- Predicting text selections

https://research.google/blog/federated-learning-with-formal-differential-privacy-guarantees/

Federated Learning

[McMahan, Moore, Ramage, Hampson, Arcas]

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
  initialize w_0
  for each round $t = 1, 2, \ldots$ **do**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (random set of m clients)
    for each client $k \in S_t$ **in parallel do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    $m_t \leftarrow \sum_{k \in S_t} n_k$
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client k*
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
  for each local epoch i from 1 to E **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(w; b)$
  return w to server

The training data never leaves the client device

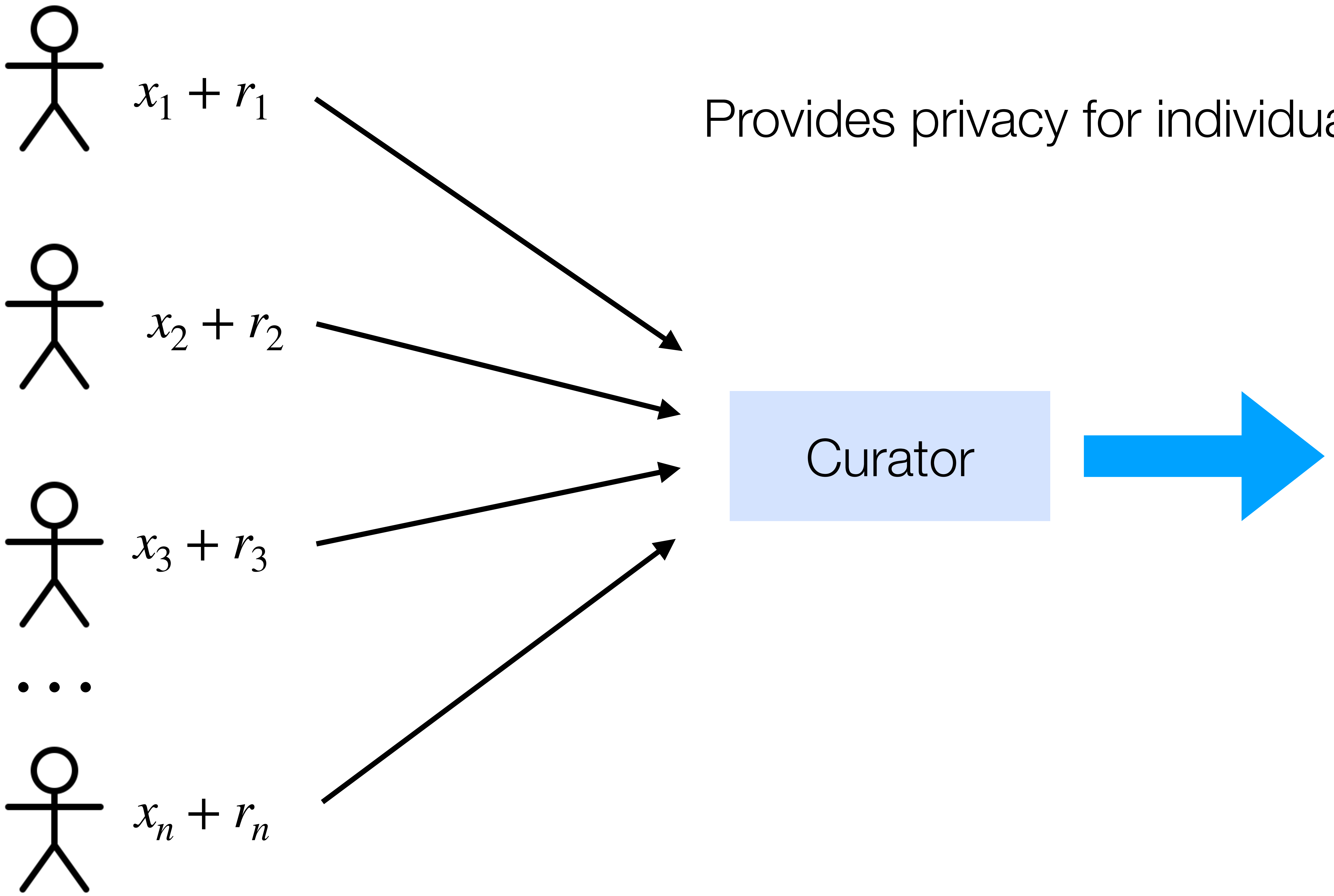
Is this enough for privacy?

No — model updates can still reveal sensitive information (student presentation)

How can we ensure the aggregator does not learn sensitive user information?

One approach: Local differential privacy

Provides privacy for individual updates



$$\sum_{i=1}^{n} (x_i + r_i)$$

Disadvantages?

Need more noise for privacy.

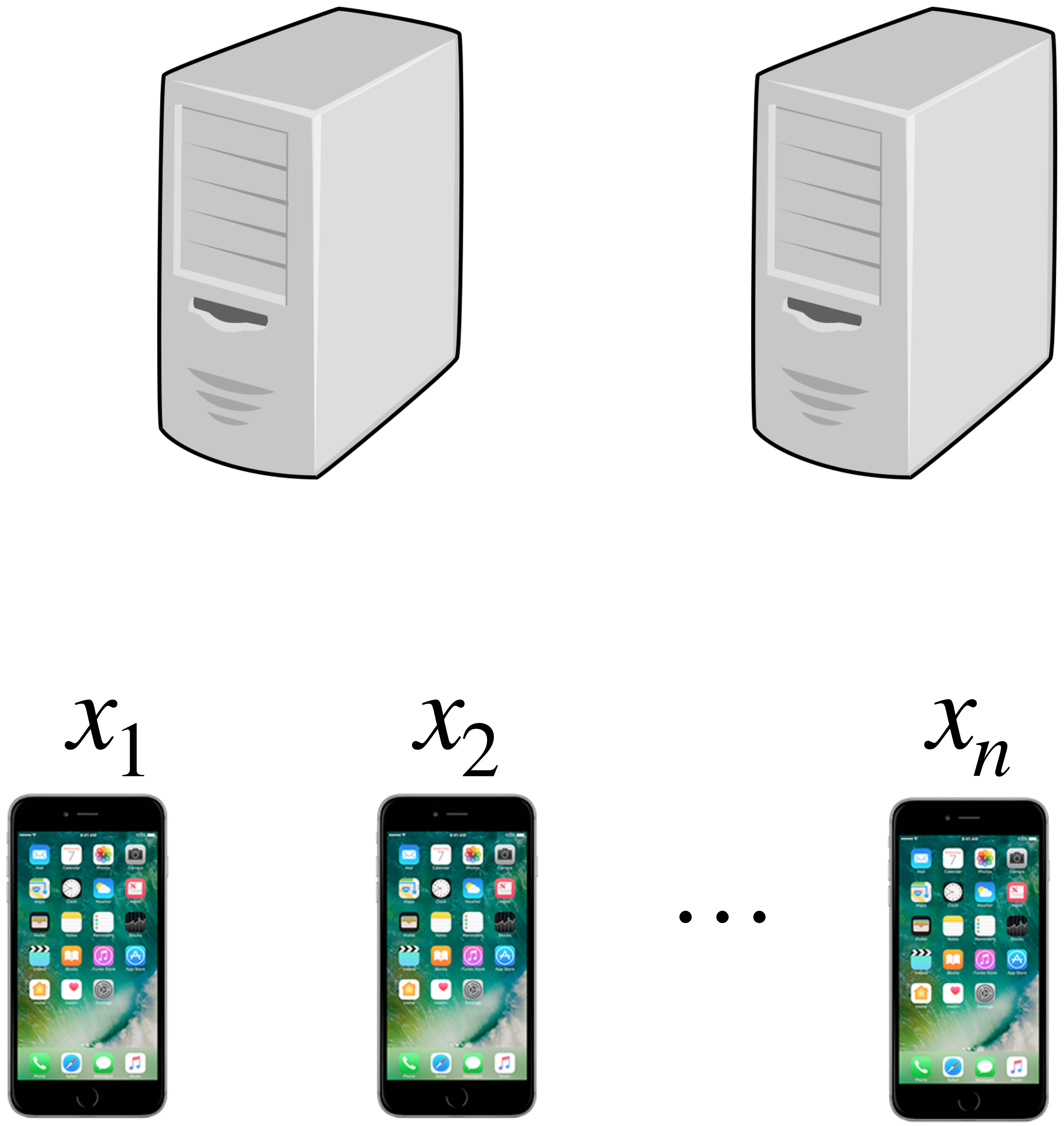
If we make ε too small, then the final statistic is not very useful

MPC + central DP is better utility vs privacy tradeoff

Another approach: two-server private aggregate statistics

- Compute a sum over client inputs
- Robustness: Malicious clients cannot overly influence sum

$$\sum_{i=1}^{n} x_i$$

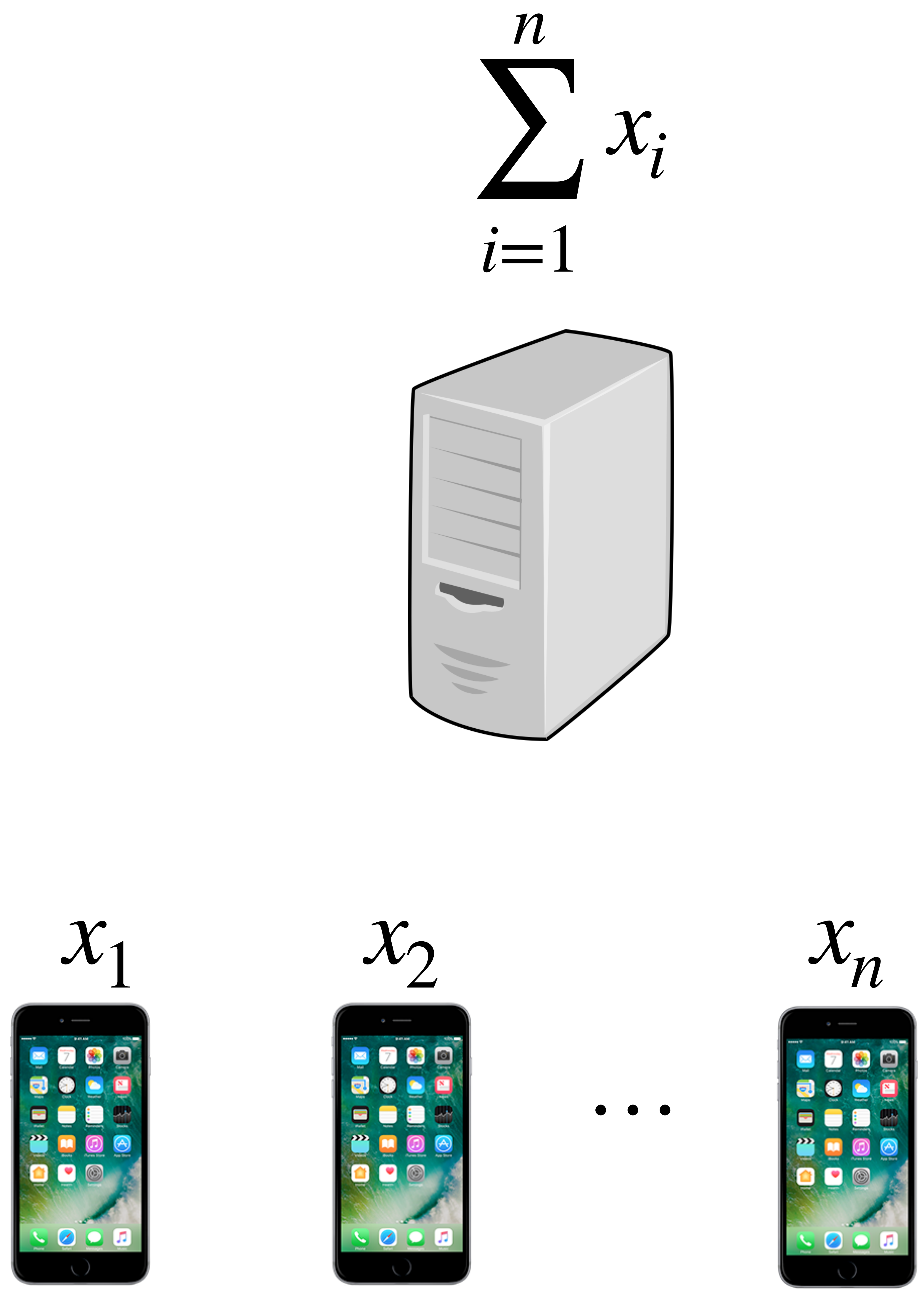


What does it mean that one user cannot have an outsize influence on the trained model?

- Popular approach: bound the L_2 norm of the update

Today: Single-server federated learning

- Goal: Server only learns the sum of all client updates, but not individual client updates



Outline

1. Federated learning
2. **Flamingo**
3. Other research topics
4. Student presentation

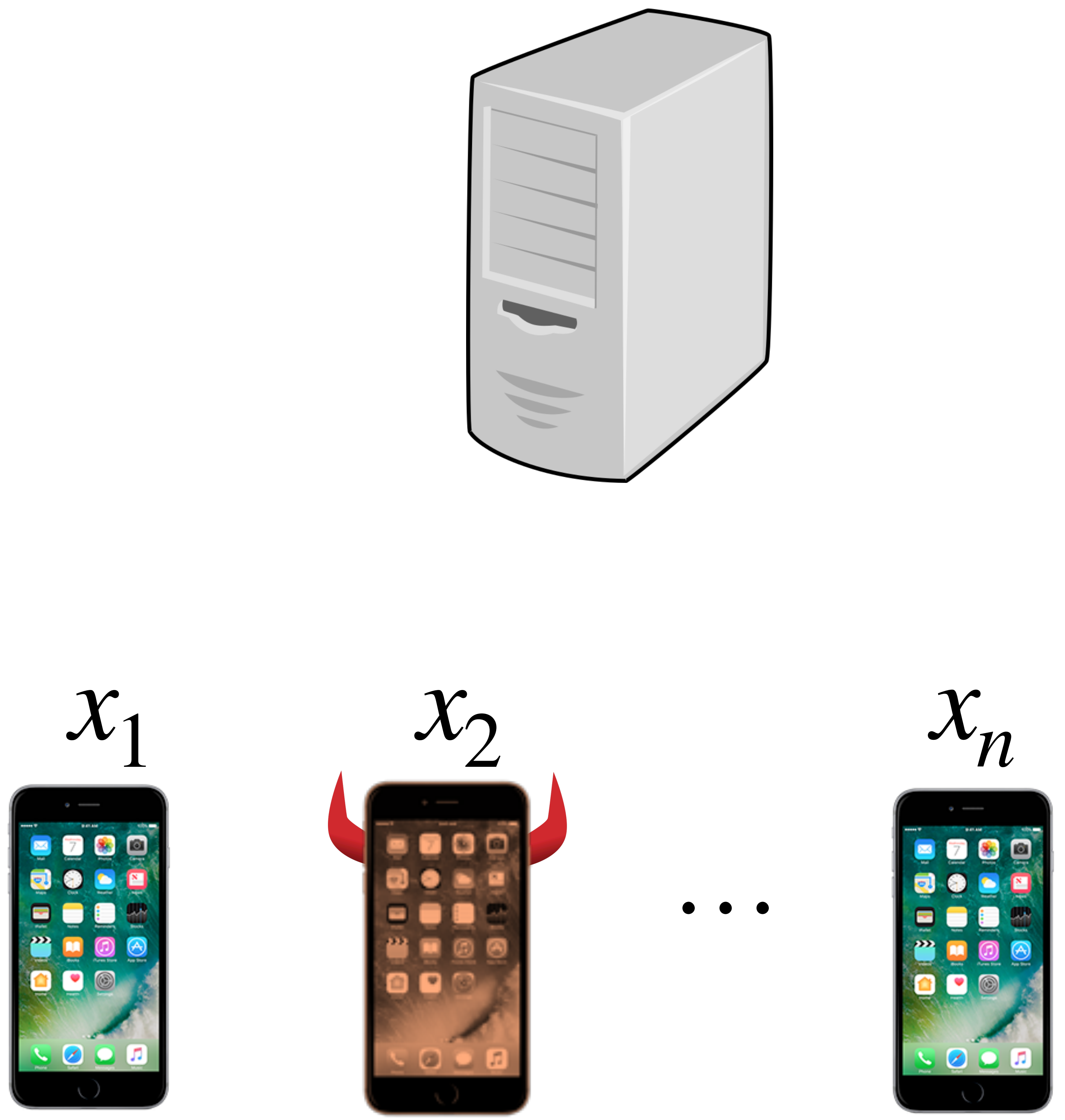
Flamingo security properties

[Ma, Woods, Angel, Polychroniadou, Rabin]

Privacy: A malicious adversary that compromises the server and some fraction of the clients only learns the sum of client inputs over at least some fraction of clients

- Individual inputs are hidden

Dropout resilience: When all parties follow the protocol, the server gets a sum of inputs from all online clients



Similarities and differences with Prio?

Background: Threshold secret sharing

Threshold secret sharing scheme across n parties where only t shares are necessary to reconstruct the original value

- $\mathsf{Share}(\alpha) \rightarrow (\alpha_1, \alpha_2, \ldots, \alpha_n)$:
- $\mathsf{Reconstruct}(\alpha_1, \alpha_2, \ldots, \alpha_t) \rightarrow \alpha$

Properties:

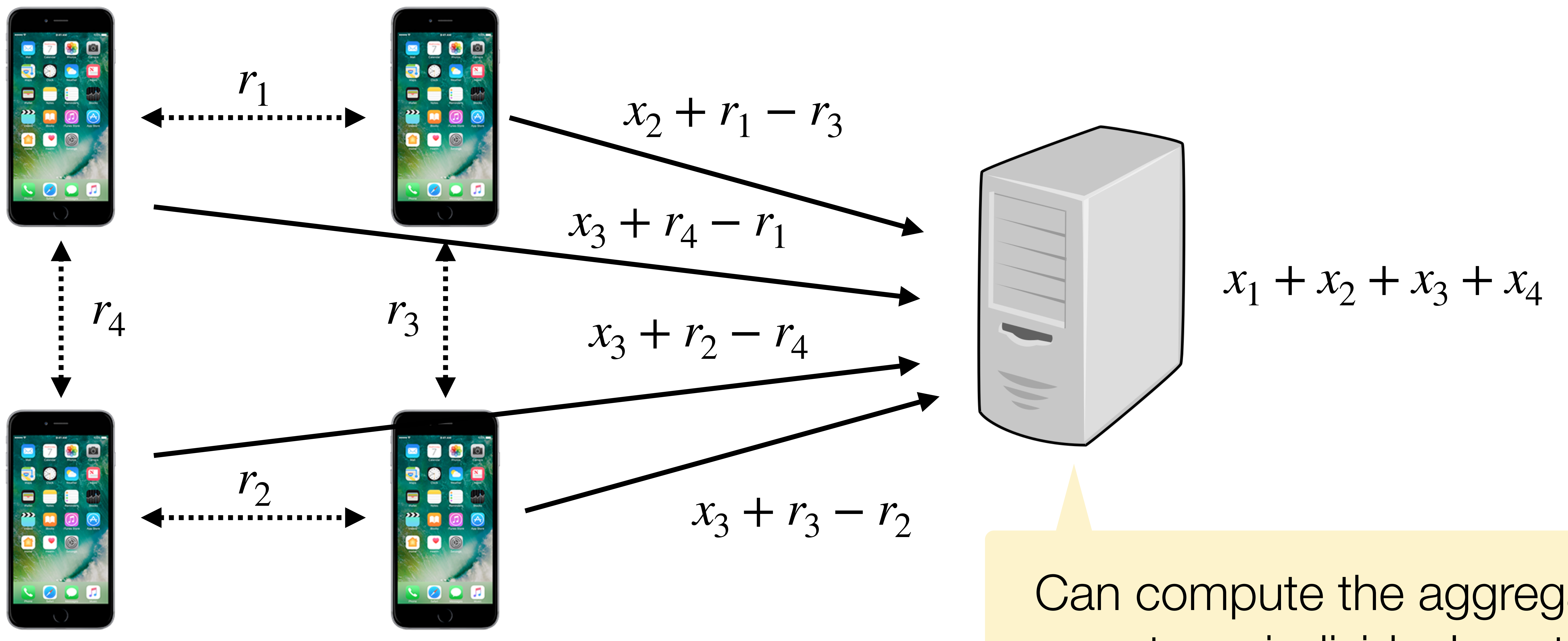
- Any group of t of the n shares can be used to reconstruct x
- Every set of $t - 1$ shares reveal nothing about x
- Can add shares locally, can multiply with communication between parties (similar to additive shares)

Background: Shamir secret sharing

Threshold secret sharing scheme across n parties where only t shares are necessary to reconstruct the original value

- Share$(\alpha) \rightarrow (\alpha_1, \alpha_2, \ldots, \alpha_n)$:
    1. Sample values $a_1, a_2, \ldots, a_t$
    2. Compute polynomial $f(x) = a_1 x + a_2 x^2 + \ldots + a_t x^t + \alpha$
    3. Output $f(i)$ for $i \in \{1,2,\ldots,n\}$
- Reconstruct$(\alpha_1, \alpha_2, \ldots, \alpha_t) \rightarrow \alpha$
    1. Interpolate the polynomial f
    2. Output $f(0)$

Idea: pairwise secrets



Can compute the aggregate, but cannot see individual contributions

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a keypair
- Each client i samples a set of “neighboring” clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a keypair
- Each client i samples a set of "neighboring" clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Step 2: Collection

- Client sends masked vector to server: $\mathbf{v}_i = \mathbf{x_i} + \sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ where $r_{i,j}$ is a shared secret between client i and j computed using one client's secret key and the other client's public key (Diffie-Hellman key exchange)
- For each offline client: the server request shares of the secret key from the client's neighbors
  For each online client: the server requests shares of m_i from the client's neighbors

Idea: pairwise masks cancel out if all clients are online (need to reconstruct pairwise masks from offline clients)

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a keypair
- Each client i samples a set of “neighboring” clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Step 2: Collection

- Client sends masked vector to server: $\mathbf{v}_i = \mathbf{x_i} + \sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ where $r_{i,j}$ is a shared secret between client i and j computed using one client’s secret key and the other client’s public key (Diffie-Hellman key exchange)
- For each offline client: the server request shares of the secret key from the client’s neighbors
  For each online client: the server requests shares of m_i from the client’s neighbors

Question: Why both the individual and the pairwise masks?

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a keypair
- Each client i samples a set of "neighboring" clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Step 2: Collection

- Client sends masked vector to server: $\mathbf{v}_i = \mathbf{x_i} + \sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ where $r_{i,j}$ is a shared secret between client i and j computed using one client's secret key and the other client's public key (Diffie-Hellman key exchange)
- For each offline client: the server request shares of the secret key from the client's neighbors
  For each online client: the server requests shares of m_i from the client's neighbors

Question: Why both the individual and the pairwise masks?
- Pairwise mask ensures that the server aggregates enough clients
- Individual mask ensures that if a message is received after the server has reconstructed the pairwise mask, it can't learn the user's input

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a keypair
- Each client i samples a set of “neighboring” clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Step 2: Collection

- Client sends masked vector to server: $\mathbf{v}_i = \mathbf{x_i} + \sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ where $r_{i,j}$ is a shared secret between client i and j computed using one client’s secret key and the other client’s public key (Diffie-Hellman key exchange)
- For each offline client: the server request shares of the secret key from the client’s neighbors

  For each online client: the server requests shares of m_i from the client’s neighbors

Question: Is this protocol secure against a malicious adversary?

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a keypair
- Each client i samples a set of "neighboring" clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Step 2: Collection

- Client sends masked vector to server: $\mathbf{v}_i = \mathbf{x_i} + \sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ where $r_{i,j}$ is a shared secret between client i and j computed using one client's secret key and the other client's public key (Diffie-Hellman key exchange)
- For each offline client: the server request shares of the secret key from the client's neighbors
  For each online client: the server requests shares of m_i from the client's neighbors

Question: Is this protocol secure against a malicious adversary?

- Server may give an inconsistent view of which clients are online/offline
- Malicious clients can return the wrong values, leading to reconstruction failure

Flamingo

Extend BBGLR protocol to:

- Provide security against a malicious adversary
- Support aggregation across multiple rounds without per-round setup

Three ideas:

1. Dropout resilience by encrypting to a small group of decryptors
2. Reusable secrets across rounds
3. Per-round graphs to handle changing client sets

Flamingo

Extend BBGLR protocol to:

- Provide security against a malicious adversary
- Support aggregation across multiple rounds without per-round setup

Three ideas:

1. **Dropout resilience by encrypting to a small group of decryptors**
2. Reusable secrets across rounds
3. Per-round graphs to handle changing client sets

Dropout resilience by encrypting to a small group of decryptors

- Group of clients (decryptors) keep secret shares of a secret decryption key
- Clients encrypt their pairwise and individual masks under the corresponding public key
- To recover the original result, they run a MPC to use their shares to decrypt ciphertext
    - Efficient, non-interactive MPC protocol for threshold decryption
- If enough of the decryptors are honest, for each client, the decryptors will decrypt one, but not both, of the two masks

$$\mathbf{v}_i = \mathbf{x_i} + \underbrace{\sum_{j \in A(i), i<j} \mathrm{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \mathrm{PRG}(r_{i,j})}_{\text{Pairwise mask}} + \underbrace{\mathrm{PRG}(m_i)}_{\text{Individual mask}}$$

Flamingo

Extend BBGLR protocol to:

- Provide security against a malicious adversary
- Support aggregation across multiple rounds without per-round setup

Three ideas:

1. Dropout resilience by encrypting to a small group of decryptors
2. **Reusable secrets across rounds**
3. Per-round graphs to handle changing client sets

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a key-pair
- Each client i samples a set of "neighboring" clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Step 2: Collection

- Client sends masked vector to server: $\mathbf{v}_i = \mathbf{x_i} + \sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ where $r_{i,j}$ is a shared secret between client i and j computed using one client's secret key and the other client's public key (Diffie-Hellman key exchange)
- For each offline client: the server request shares of the secret key from the client's neighbors
  For each online client: the server requests shares of m_i from the client's neighbors

Question: What goes wrong if you re-use the same secret across rounds?

Starting point: BBGLR protocol

Step 1: Setup

- Each client samples a key-pair
- Each client i samples a set of "neighboring" clients A_i and sends the neighboring clients its public key, forming a graph
- Each client i uses Shamir secret sharing to split its secret key and a random value m_i across neighboring clients

Step 2: Collection

- Client sends masked vector to server: $\mathbf{v}_i = \mathbf{x_i} + \sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ where $r_{i,j}$ is a shared secret between client i and j computed using one client's secret key and the other client's public key (Diffie-Hellman key exchange)
- For each offline client: the server request shares of the secret key from the client's neighbors
  For each online client: the server requests shares of m_i from the client's neighbors

Question: What goes wrong if you re-use the same secret across rounds?
- If a client is online one round and offline the next, the server can learn both the pairwise and individual mask

Reusable secrets across rounds

The client uses the shared secret $r_{i,j}$ as a PRG to compute a per-round pairwise mask

BBGLR

$$\mathbf{v}_i = \mathbf{x_i} + \underbrace{\sum_{j \in A(i), i<j} \mathrm{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \mathrm{PRG}(r_{i,j})}_{\text{Pairwise mask}} + \underbrace{\mathrm{PRG}(m_i)}_{\text{Individual mask}}$$

Flamingo

$$\mathbf{v}_i = \mathbf{x_i} + \underbrace{\sum_{j \in A(i), i<j} \mathrm{PRG}(r_{i,j}, t) - \sum_{j \in A(i), i>j} \mathrm{PRG}(r_{i,j}, t)}_{\text{Pairwise mask}} + \underbrace{\mathrm{PRG}(m_i)}_{\text{Individual mask}}$$

Round t

Flamingo

Extend BBGLR protocol to:

- Provide security against a malicious adversary
- Support aggregation across multiple rounds without per-round setup

Three ideas:

1. Dropout resilience by encrypting to a small group of decryptors
2. Reusable secrets across rounds
3. **Per-round graphs to handle changing client sets**

Reusable secrets across rounds

- BBGLR uses a sparse graph (for pairwise secrets) to minimize communication between clients
- But across rounds, clients may go offline —> don't want to set up a new graph every time
- One approach: Make one graph at setup time, and then use the subgraph for the participating clients in each subsequent round

Reusable secrets across rounds

- BBGLR uses a sparse graph (for pairwise secrets) to minimize communication between clients
- But across rounds, clients may go offline —> don't want to set up a new graph every time
- One approach: Make one graph at setup time, and then use the subgraph for the participating clients in each subsequent round
  - Problem: If the graph is fairly sparse, and so if clients go online it might not be connected and may have isolated nodes (hurting privacy)
  - Problem: If the graph is more dense, then the communication overheads are high

Reusable secrets across rounds

- BBGLR uses a sparse graph (for pairwise secrets) to minimize communication between clients
- But across rounds, clients may go offline —> don't want to set up a new graph every time
- One approach: Make one graph at setup time, and then use the subgraph for the participating clients in each subsequent round
  - Problem: If the graph is fairly sparse, and so if clients go online it might not be connected and may have isolated nodes (hurting privacy)
  - Problem: If the graph is more dense, then the communication overheads are high
- Idea: client can use random seed along with knowledge of which clients are participating in the round to compute its set of neighbors
  - Does not have to materialize the entire graph

Outline

1. Federated learning
2. Flamingo
3. **Other research topics**
4. Student presentation

Zero-knowledge proofs

Informally: Prove that some statement is true without revealing the evidence

More formally: For languages $\mathscr{L}$ in NP with instance x and witness w, prove that $x \in \mathscr{L}$ without revealing w

Applications:

- Cryptocurrencies and blockchains
- Anonymous credentials (age verification, CAPTCHA alternative)
- Middleboxes that enforce properties on encrypted traffic [Zero-knowledge middleboxes]
- Edited image is derived from a photo taken by a certified camera [VerITAS]
- Encrypted authentication logging [Larch]
- … and more

Homomorphic encryption

- Perform arbitrary computation on encrypted data
- Most efficient for circuits with low multiplicative depth
- Possible to perform computations with unbounded depth, but at a high concrete cost
- Applications to:
  - Searching on data (e.g., Tiptoe)
  - Inference where the client's input remains hidden
  - … and more

Secure inference

- How can you query a model hosted at a server without the server seeing your query?
- Challenge: large scale, operations that are not “crypto-friendly”
- Common approach: use a blend of multi-party computation and homomorphic encryption (Bolt, Delphi, Gazelle)
- Moving forward:
  - Can we build better cryptographic tools tailored to transformers?
  - Are there ways we can better co-design transformer inference and cryptographic tools?
  - Are there opportunities to use hardware to accelerate cryptographic operations?

Next steps for security + cryptography at Stanford

Courses:
- CS 251: Cryptocurrencies and blockchain technologies (fall)
- CS 255: Introduction to Cryptography (winter)
- CS 258: Quantum Cryptography (fall)
- CS 329T: Trustworthy Machine Learning (fall)
- CS 355: Advanced Topics in Cryptography (spring)
- CS 356: Topics in Computer and Network Security (fall)
- CS 357S: Formal Methods for Computer Systems (winter)

Events:
- Security lunch (Wednesday @12PM, CoDa E160): https://securitylunch.stanford.edu/
- Security seminar (some Thursdays @4PM): https://crypto.stanford.edu/seclab/sem.html

If you're interested in research / continuing your course project, reach out!

Week 10: Final presentations

12/2, 12/4: 9 minutes for presentation, 2 minutes for questions / transitioning to next group

12/2: Final project reports due

Outline

1. Federated learning
2. Flamingo
3. Other research topics
4. **Student presentation**

References

Bell, James Henry, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. "Secure single-server aggregation with (poly) logarithmic overhead." In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pp. 1253-1269. 2020.

Datta, Trisha, Binyi Chen, and Dan Boneh. "VerITAS: Verifying image transformations at scale." In *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 4606-4623. IEEE, 2025.

Dauterman, Emma, Danny Lin, Henry Corrigan-Gibbs, and David Mazières. "Accountable authentication with privacy protection: The Larch system for universal login." In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 81-98. 2023.

Grubbs, Paul, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. "{Zero-Knowledge} middleboxes." In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 4255-4272. 2022.

Juvekar, Chiraag, Vinod Vaikuntanathan, and Anantha Chandrakasan. "{GAZELLE}: A low latency framework for secure neural network inference." In *27th USENIX security symposium (USENIX security 18)*, pp. 1651-1669. 2018.

Ma, Yiping, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning." In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 477-496. IEEE, 2023.

McMahan, Brendan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-efficient learning of deep networks from decentralized data." In *Artificial intelligence and statistics*, pp. 1273-1282. PMLR, 2017.

Mishra, Pratyush, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. "Delphi: A cryptographic inference system for neural networks." In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, pp. 27-30. 2020.

Pang, Qi, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. "Bolt: Privacy-preserving, accurate and efficient inference for transformers." In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 4753-4771. IEEE, 2024.

https://research.google/blog/federated-learning-collaborative-machine-learning-without-centralized-training-data/

https://research.google/blog/federated-learning-with-formal-differential-privacy-guarantees/