Flamingo: Multi-Round Single-Server Secure Aggregation with Applications to Private Federated Learning

Yiping Ma¹ Jess Woods¹ Sebastian Angel¹,² Antigoni Polychroniadou³ Tal Rabin¹

¹University of Pennsylvania
²Microsoft Research
³J.P. Morgan AI Research & AlgoCRYPT CoE
Data-driven applications nowadays

Service providers collect and analyze user data in order to provide customized functionalities.
Data-driven applications nowadays

Simply put, to protect users at scale, we rely on machine learning powered by user feedback to catch spam and help us identify patterns in large data sets—making it easier to adapt quickly to ever-changing spam tactics. Gmail employs a number of AI-driven filters that determine what gets marked as spam. These filters look at a variety of signals, including characteristics of the IP address, domains/subdomains, whether bulk senders are authenticated, and user input. User feedback, such as when a user marks a certain email as spam or signals they want a sender’s emails in their inbox, is key to this filtering process, and our filters learn from user actions.
Building a machine learning model

Step 1. Collect data from many users.

(“hacked”, spam)  ("malicious", spam)  ("hotel", not spam)
Building a machine learning model

Step 2. Run a training algorithm on the collected data.
Building a machine learning model

Step 3. Deploy the model in their services.
Centralized vs. decentralized training

Centralized

\[ \text{Train} \left( \begin{array}{c}
\text{ (“hacked”, spam)} \\
\text{ (“hotel”, not spam)} \\
\text{ (“malicious”, spam)} 
\end{array} \right) \]

Data never leaves user devices!

Decentralized

“Federated learning” [McMahan et al. in 2016]

Many clients (users) collaboratively train a model under the orchestration of a central server (service provider).
Centralized vs. decentralized training

### Centralized

- **Train**
  - (“hacked”, spam)
  - (“hotel”, not spam)
  - (“malicious”, spam)

### Decentralized

- **Update**
  - (“hacked”, spam)
- **Update**
  - (“malicious”, spam)
- **Update**
  - (“hotel”, not spam)

Local weights
Centralized vs. decentralized training

**Centralized**

Centralized training can be represented as:

\[
\text{Train} = \begin{cases} 
(\text{"hacked"}, \text{spam}) \\
(\text{"hotel"}, \text{not spam}) \\
(\text{"malicious"}, \text{spam})
\end{cases}
\]

**Decentralized**

Decentralized training involves updates from multiple devices:

- Update \((\text{"hacked"}, \text{spam})\)
- Update \((\text{"malicious"}, \text{spam})\)
- Update \((\text{"hotel"}, \text{not spam})\)

Global weights are then aggregated from these updates.
Centralized vs. decentralized training

Centralized

= \text{Train}\left(\begin{array}{l}
\text{("hacked", spam)} \\
\text{("hotel", not spam)} \\
\text{("malicious", spam)} \\
\text{("apple", not spam)} \\
\text{("dog", not spam)} \\
\text{("lottery", spam)} \\
\text{("random", spam)}
\end{array}\right)

Decentralized

= ++

A few hundreds to a few thousands of clients
Centralized vs. decentralized training

Centralized

Centralized training:

= \text{Train}\left(\begin{array}{c}
("hacked", \text{spam}) \\
("hotel", \text{not spam}) \\
("malicious", \text{spam}) \\
("apple", \text{not spam}) \\
("dog", \text{not spam}) \\
("lottery", \text{spam}) \\
("random", \text{spam})
\end{array}\right)

Decentralized

Decentralized training:

= + +
Centralized vs. decentralized training

Centralized

= \text{Train}\left(\begin{array}{l}
(\text{“hacked”, spam})\\(\text{“hotel”, not spam})\\(\text{“malicious”, spam})\\(\text{“apple”, not spam})\\(\text{“dog”, not spam})\\(\text{“lottery”, spam})\\(\text{“random”, spam})
\end{array}\right)

Decentralized

= + +
Federated learning: steps forward

Are we done?

• Inference attacks
  Infer information about individual data from local weights [Zhu et al. 2019]

• Poisoning attacks
  Contribute bogus training data [Biggio et al. 2012] or bogus model weights to undermine the performance of trained models [Bagdasaryan et al. 2018]
Federated learning: steps forward

Are we done?

- **Inference attacks:**
  Infer information about individual data from local weights [Zhu et al. 2019]

- **Poisoning attacks:**
  Contribute bogus training data [Biggio et al. 2012] or bogus model weights to undermine the performance of trained models [Bagdasaryan et al. 2018]
Secure aggregation for federated learning

• Secure aggregation (A special case of MPC [Yao 1986])

Only learns $x_1 + x_2 + x_3$

Many works under different communication models, cryptographic assumptions, etc.

• Secret sharing
  [KRKR 2020], [DSQG+ 2022], ...

• Threshold homomorphic encryption
  [SGA 2021], [SHYL+ 2022], ...

• Pairwise masking
  [BIKM+ 2017], [BBGLR 2020], ...

• Other styles
  [SSVR+ 2022], [GPSBB 2022], ...
Secure aggregation for federated learning

- Secure aggregation (A special case of MPC [Yao 1986])

- Only learns $x_1 + x_2 + x_3$

- Many works under different communication models, cryptographic assumptions, etc.

Concrete efficiency but not asymptotics

- Secret sharing [KRKR 2020], [DSQG+ 2022], …
- Threshold homomorphic encryption [SGA 2021], [SHYL+ 2022], …
- Pairwise masking [BIKM+ 2017], [BBGLR 2020], …
- Other styles [SSVR+ 2022], [GPSBB 2022], …
Federated learning has complex setting

• From the federation side—restricted clients (mobile devices)
  • Limited computation power
  • Unstable network connection

• From the machine learning side—large parameters
  • Inputs: model weights, e.g., ~500K in popular models for CIFAR100
  • Participants: 100-5000 per iteration
  • Training: many iterations, e.g., ~300 for CIFAR100
Federated learning has complex setting

• From the federation side—restricted clients (mobile devices)
  • Limited computation power
  • Unstable network connection

      Lightweight client computation

      Tolerate dropouts at any point

• From the machine learning side—large parameters
  • Inputs: model weights, e.g., ~500K in popular models for CIFAR100
  • Participants: 100-5000 per iteration
  • Training: many iterations, e.g., ~300 for CIFAR100
Federated learning has complex setting

• From the federation side—restricted clients (mobile devices)
  • Limited computation power
  • Unstable network connection

• From the machine learning side—large parameters
  • Inputs: model weights, e.g., ~500K in popular models for CIFAR100
  • Participants: 100-5000 per iteration
  • Training: many iterations, e.g., ~300 for CIFAR100

Reasonable efficiency
Prior designs are not the best fit for a full training

- From the federation side—restricted clients (mobile devices)
  - Limited computation power
  - Unstable network connection

- From the machine learning side—large parameters
  - Inputs: model weights, e.g., ~500K in popular models for CIFAR100
  - Participants: 100-5000 per iteration
  - Training: many iterations, e.g., ~300 for CIFAR100

One summation: multiple round trips, some of which are expensive
Having fewer round trips is important

• Reduce bias and improve quality

• Reduce run time

Will discuss in evaluation section why round trips matter a lot
We propose Flamingo

• From the federation side—restricted clients (mobile devices)
  • Limited computation power
  • Unstable network connection

  Lightweight client computation
  Tolerate dropouts at any point

• From the machine learning side—large parameters
  • Inputs: model weights, e.g., ~500K in popular models for CIFAR100
  • Participants: 100-5000 per iteration
  • Training: many iterations, e.g., ~300 for CIFAR100

  Can practically run for a full training session

Same threat model as in prior work: a malicious adversary controlling the server and a subset of the clients
Flamingo has two key ideas

• A fault-tolerant private sum protocol based on pairwise secrets and threshold decryption

• A way to reuse pairwise secrets over many iterations
A fault-tolerant private sum protocol

Pairwise secrets

BIKM+ 2017, BBGLR 2020

Take some cost to set them up

\[ v_1 = x_1 + \text{PRG}(s_{12}) + \text{PRG}(s_{13}) \]
A fault-tolerant private sum protocol

Pairwise secrets

BIKM+ 2017, BBGLR 2020

Efficient despite large inputs

\[
\begin{align*}
\overrightarrow{v_1} &= \overrightarrow{x_1} + \text{PRG}(s_{12}) + \text{PRG}(s_{13}) \\
\overrightarrow{v_2} &= \overrightarrow{x_2} - \text{PRG}(s_{12}) + \text{PRG}(s_{23}) \\
\overrightarrow{v_3} &= \overrightarrow{x_3} - \text{PRG}(s_{13}) - \text{PRG}(s_{23})
\end{align*}
\]

\[
\sum_{i=1}^{3} \overrightarrow{v_i} = \sum_{i=1}^{3} \overrightarrow{x_i}
\]
A fault-tolerant private sum protocol

Pairwise secrets

\[ \overrightarrow{v_1} = \overrightarrow{x_1} + \text{PRG}(s_{12}) + \text{PRG}(s_{13}) \]

\[ \overrightarrow{v_2} = \overrightarrow{x_2} - \text{PRG}(s_{12}) + \text{PRG}(s_{23}) \]

Went offline...
A fault-tolerant private sum protocol

Threshold decryption

How are they chosen? How is $SK$ shared among them?
A fault-tolerant private sum protocol

Threshold decryption

\[
\begin{align*}
\vec{v}_1 &= \vec{x}_1 + \text{PRG}(s_{12}) + \text{PRG}(s_{13}) \\
\vec{v}_2 &= \vec{x}_2 - \text{PRG}(s_{12}) + \text{PRG}(s_{23}) \\
\vec{v}_3 &= \vec{x}_3 - \text{PRG}(s_{13}) - \text{PRG}(s_{23})
\end{align*}
\]

\[
\vec{v}_1 + \vec{v}_2 - \text{PRG}(s_{13}) - \text{PRG}(s_{23}) = \vec{x}_1 + \vec{x}_2
\]
A fault-tolerant private sum protocol

Threshold decryption

Recovery is lightweight
Reusing the secrets

Essentially OTP

Iteration $t$: $s_{12}^t = \text{PRF}(s_{12}, t)$

Simple idea, but cannot work for [BBGLR 2020] due to a crucial design difference for fault tolerance

$\overrightarrow{v}_1 = \overrightarrow{x}_1 + \text{PRG}(s_{12}^t) + \text{PRG}(s_{13}^t)$

Enc($PK$, $s_{12}^t$)

Enc($PK$, $s_{13}^t$)
With the two key ideas →

Do the costly setup once,  
and run the lightweight sum many times
More details in the paper

• How **decryptors** work
  • Selection
  • Sharing of **SK**
  • Switching decryptors over time

• How **setup** is done

• How to achieve **malicious security**

• Efficient instantiation of cryptographic primitives, system-level optimizations
Evaluation results

• What is the right factor to look at?
  • Computation cost was the focus: [BIKM+ 2017] \rightarrow [BBGLR 2020]
  • When computation is made cheap, what matters is the “waiting time”
Evaluation results

• What is the right factor to look at?
  • Computation cost was the focus: [BIKM+ 2017] → [BBGLR 2020]
  • When computation is made cheap, what matters is the “waiting time”
Evaluation results

• Feasibility of training a neural network on CIFAR100

We used ABIDES simulator: https://github.com/abides-sim/abides
Summary

• This work: A secure aggregation system that handles real-world federated training tasks
• Many interesting future directions
  • Validation of client inputs
  • Stronger security, e.g., adaptive adversary

Code available at https://github.com/eniac/flamingo

Thanks!
Backup Slides
Malicious security

\[ \text{Enc}(PK, s_{12}) \]
\[ \text{Enc}(PK, s_{13}) \]
\[ \text{Enc}(PK, s_{23}) \]

Alice

\[ \overrightarrow{v_1} \]

Bob

\[ \overrightarrow{v_2} \]
Malicious security

Alice

Bob

Charlie

Decryptors

$\text{Enc}(PK, s_{12})$

$\text{Enc}(PK, s_{13})$

$s_{12}$

$s_{13}$

$s_{23}$

$\overrightarrow{x_1}$

$\overrightarrow{x_2}$

$\overrightarrow{x_3}$

$SK_1$

$SK_2$

$SK_3$
A cross-check round

Key idea:
Honest decryptors agree on what to decrypt
A fault-tolerant sum with malicious security

\[ x_1, x_2, x_3 \]

Waiting...

Process...

\[ \overrightarrow{v_1} + \overrightarrow{v_2} \]

Enc(PK, s_{12}^t)

Enc(PK, s_{13}^t)

Waiting...

Process...

Waiting...

Process...

\[ \overrightarrow{v_1} + \overrightarrow{v_2} \]

- PRG(s_{12}^t)

- PRG(s_{13}^t)

Partial decryptions

\[ \overrightarrow{x_1} + \overrightarrow{x_2} \]