# *Dordis*

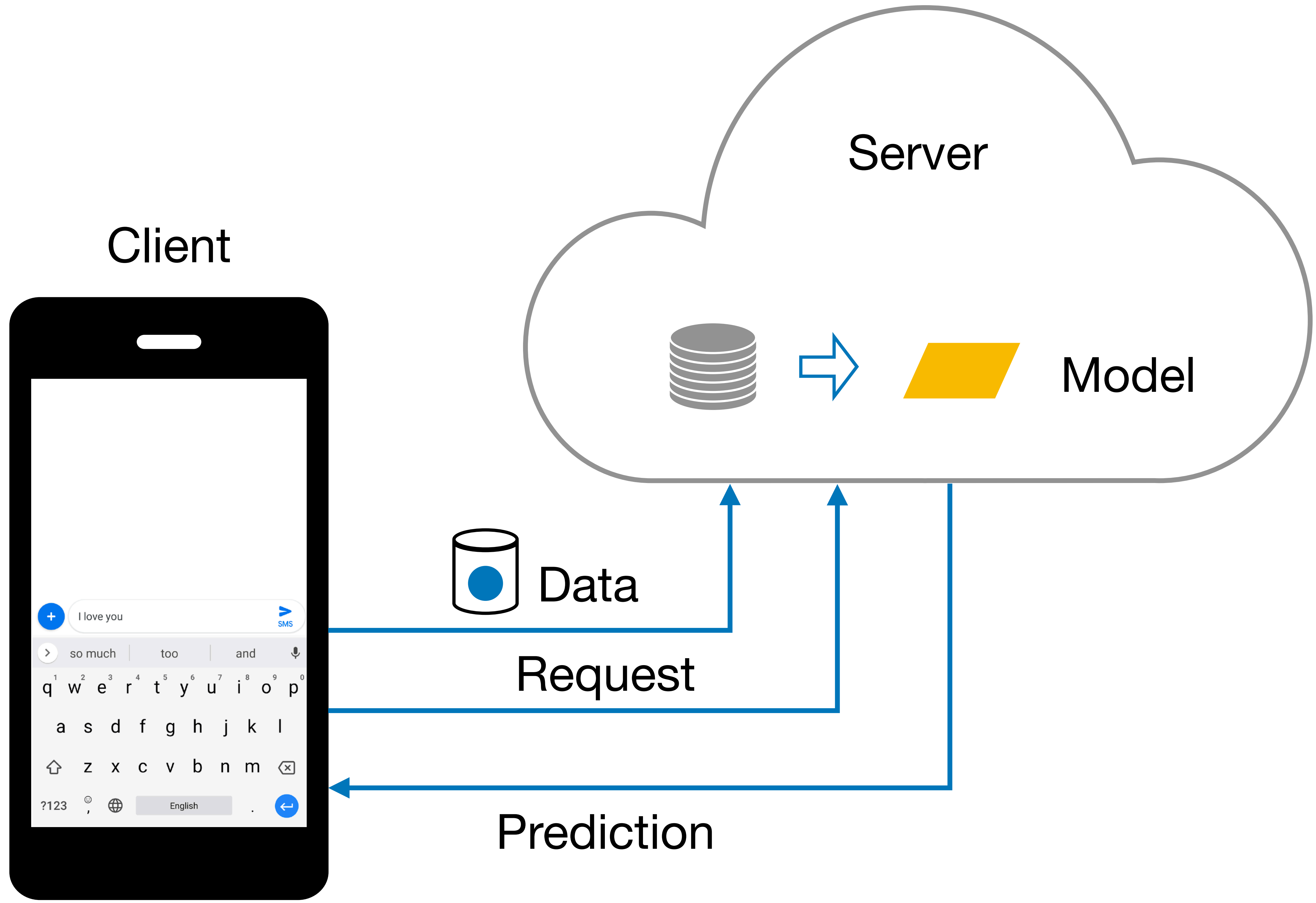
## *Efficient Federated Learning with Dropout-Resilient Differential Privacy*

**Zhifeng Jiang,**
Wei Wang,
Ruichuan Chen

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

NOKIA BELL LABS

# Centralized learning

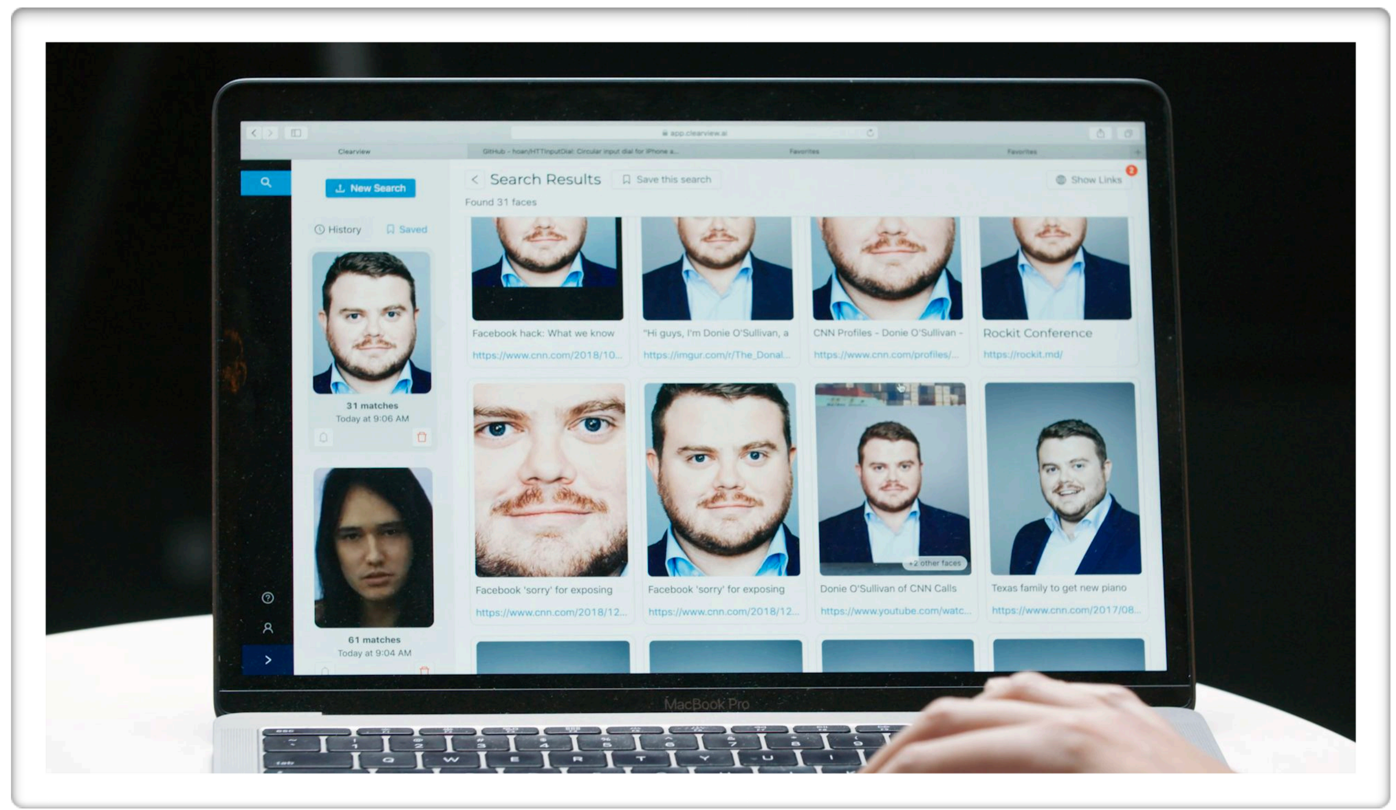


# Centralized learning hurts privacy

Data breaches…

Forbes

**Clearview AI, The Company Whose Database Has Amassed 3 Billion Photos, Hacked**



# Centralized learning hurts privacy

Data breaches…

Forbes

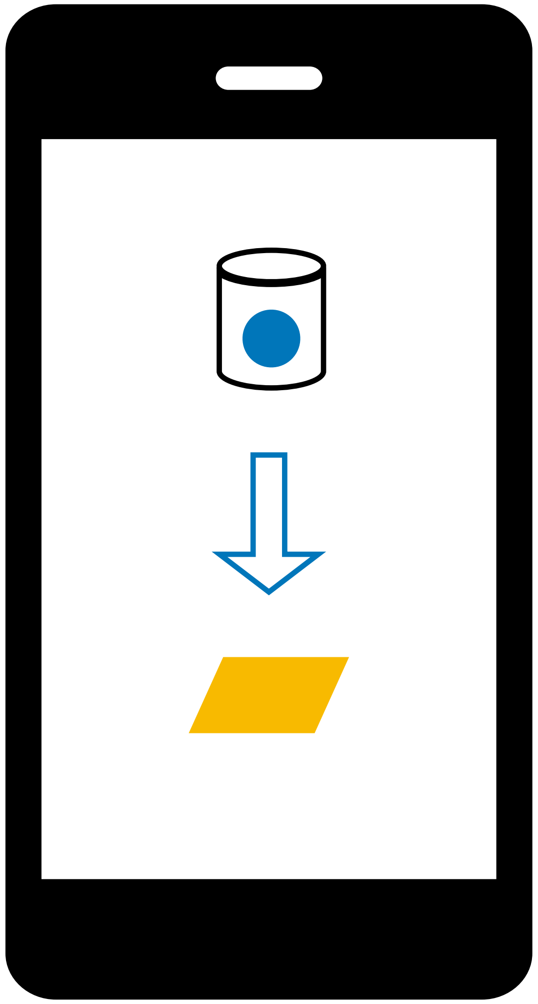
Clearview AI, The Company Whose Database Has Amassed 3 Billion Photos, Hacked

Potential abuse…

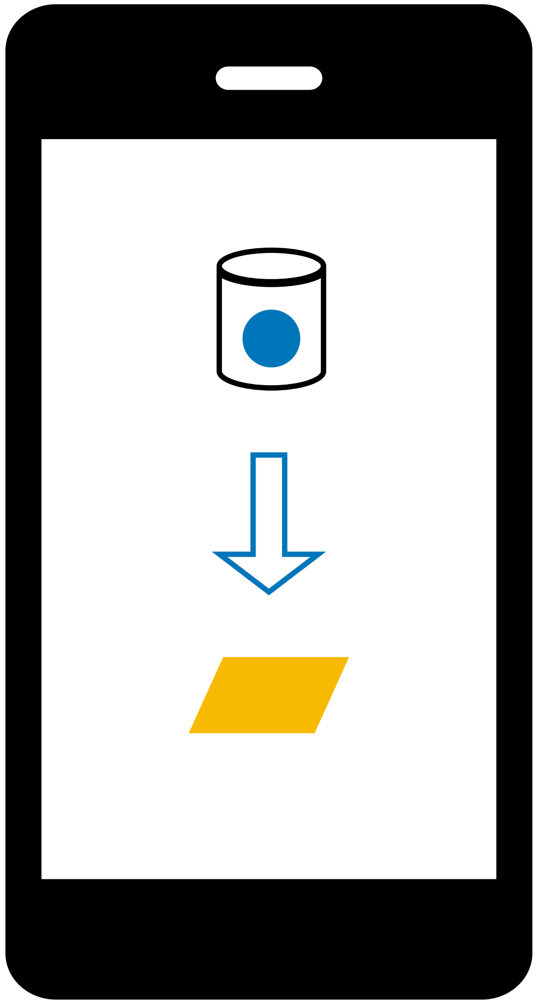
theguardian

Facebook halts use of WhatsApp data for advertising in Europe

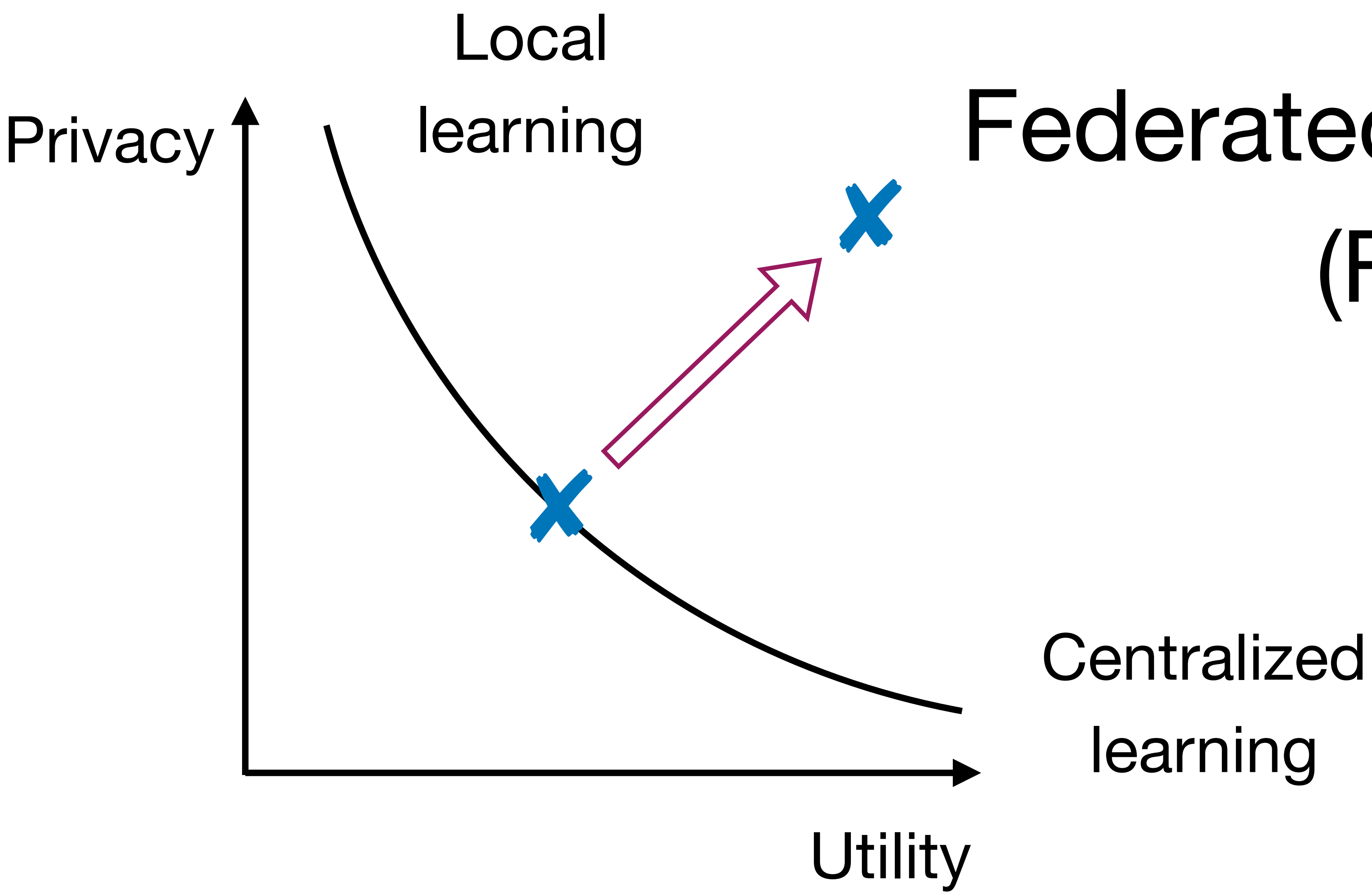
# Local learning



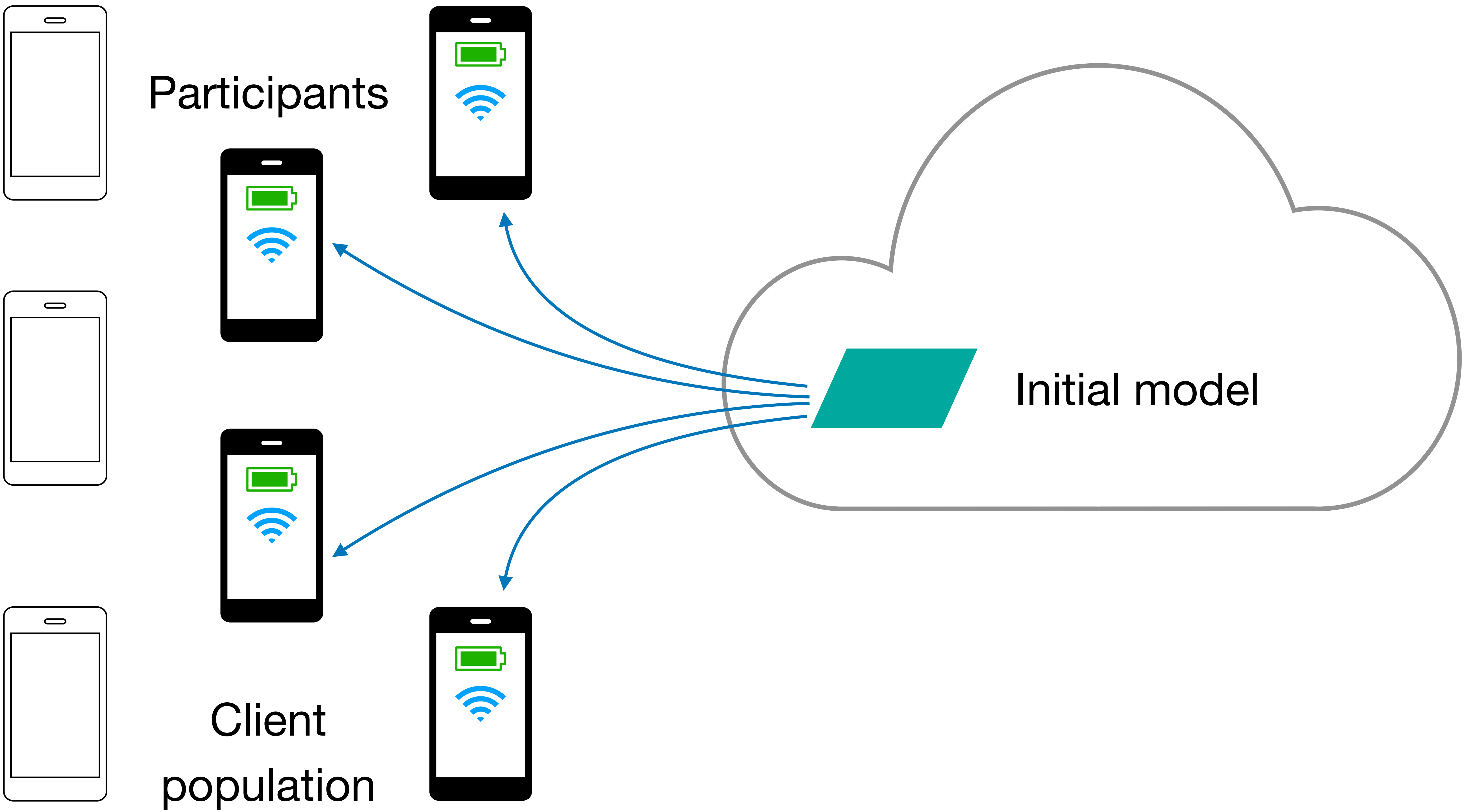
# Local learning suffers from low utility



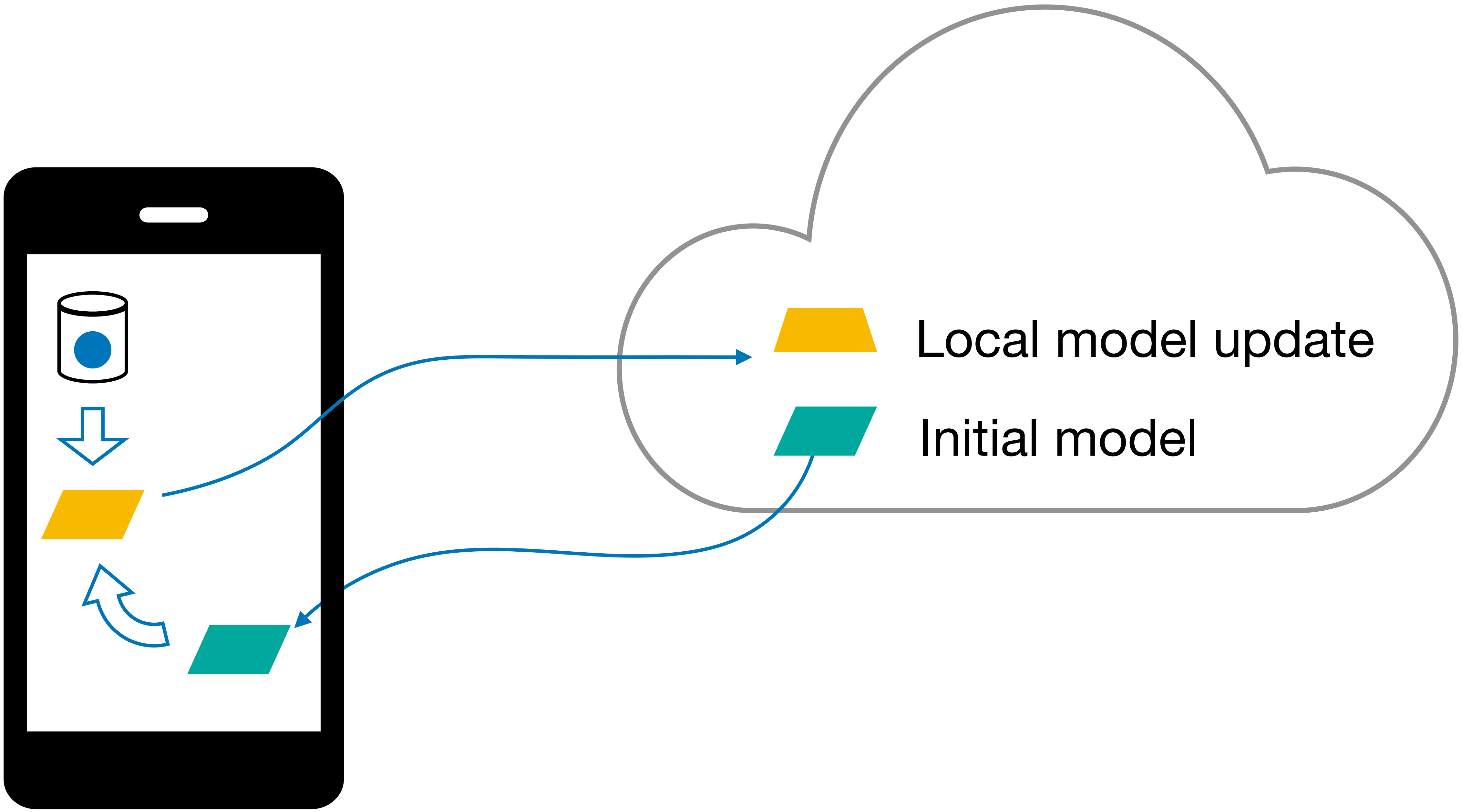
# Federated Learning (FL)



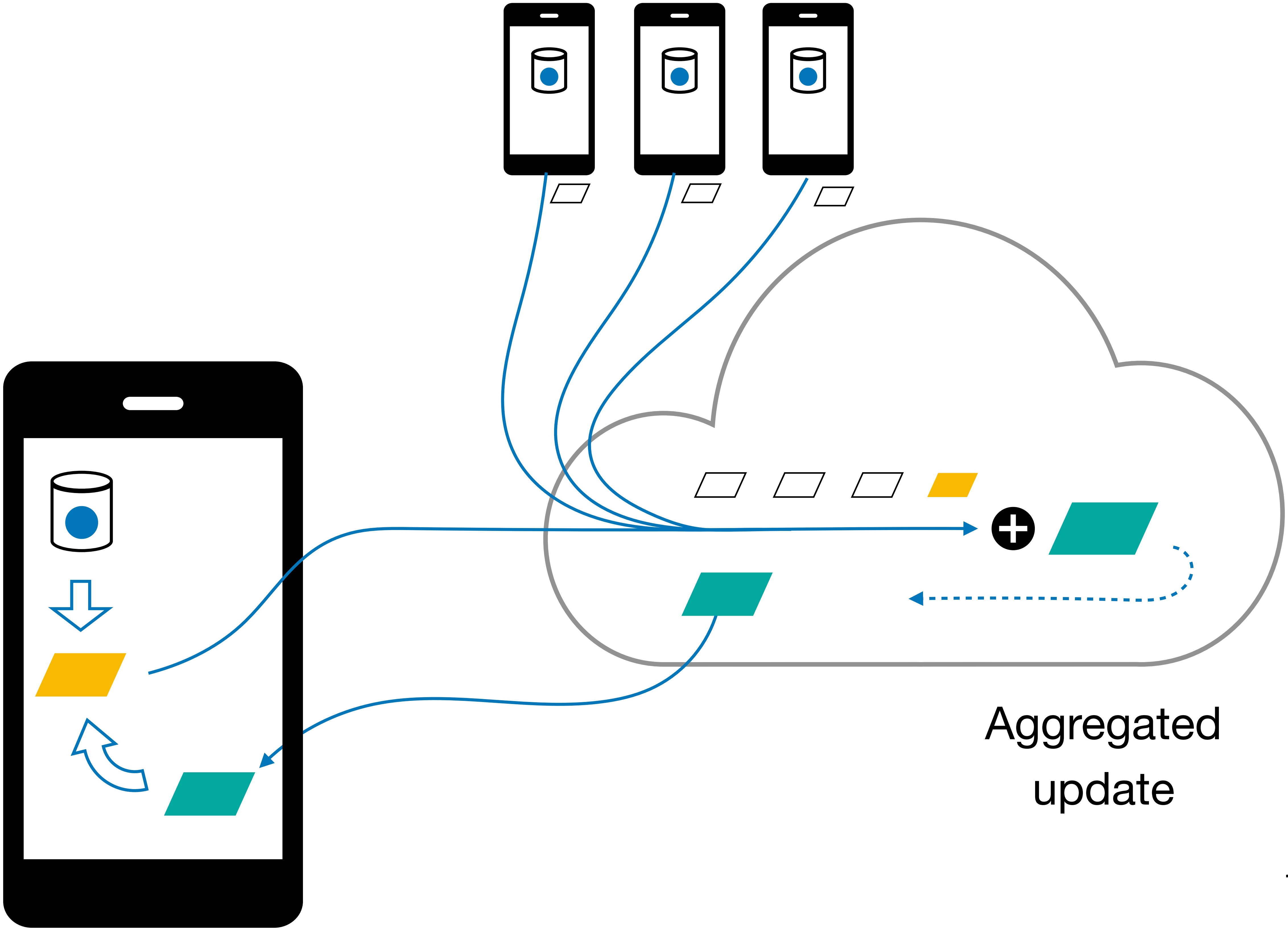
# FL Step 1: Participant Selection



# FL Step 2: Local Training

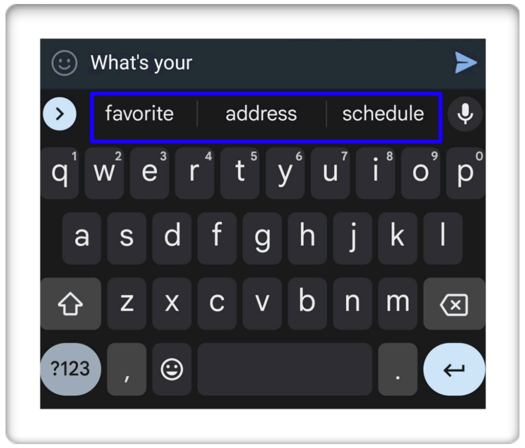


# FL Step 3: Model Aggregation



# FL Applications

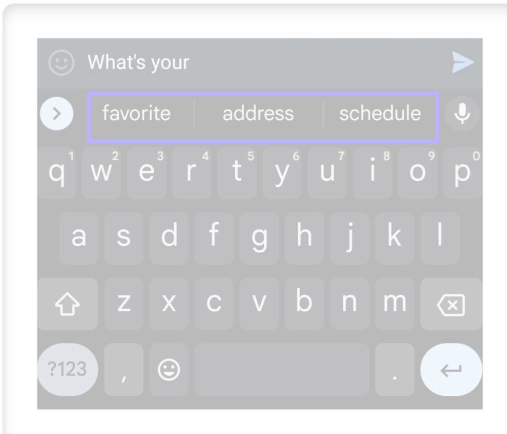
Mobile




Google's Keyboard

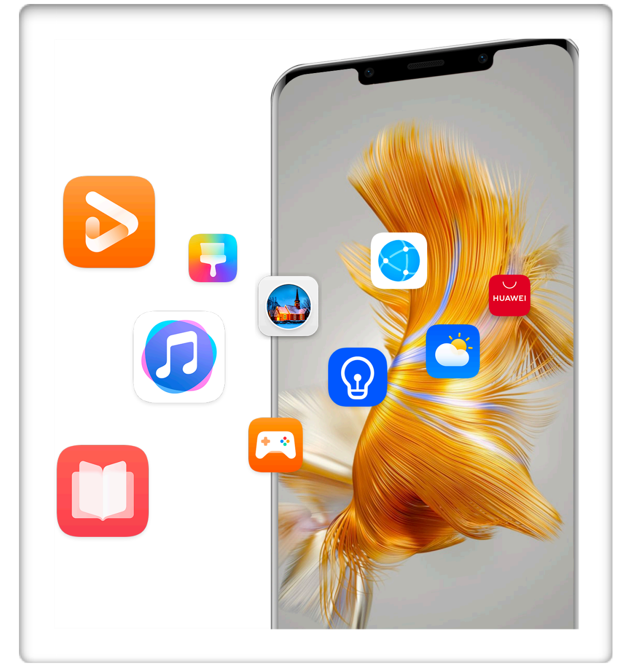
# FL Applications

Mobile

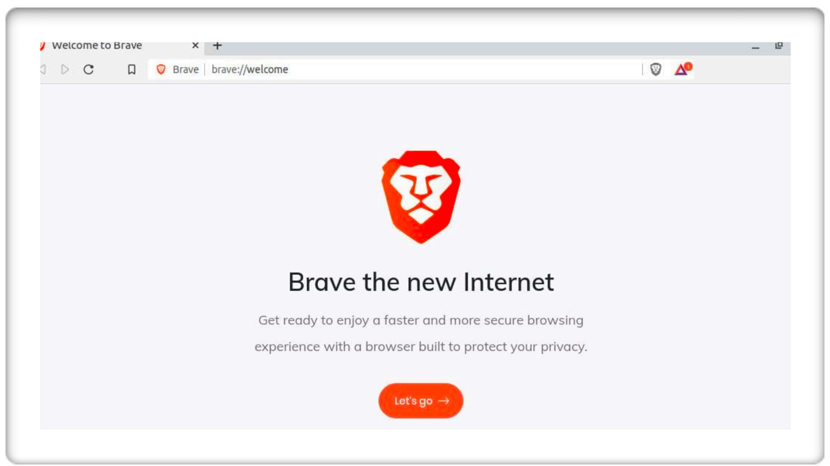



Google's Keyboard

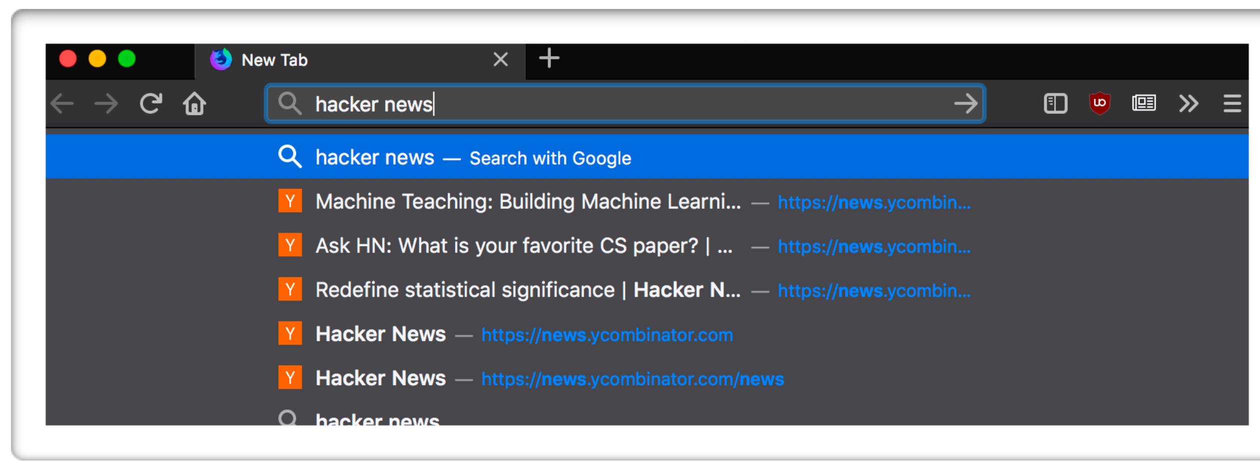
Apple's speaker recognition



Huawei's ads recommendation

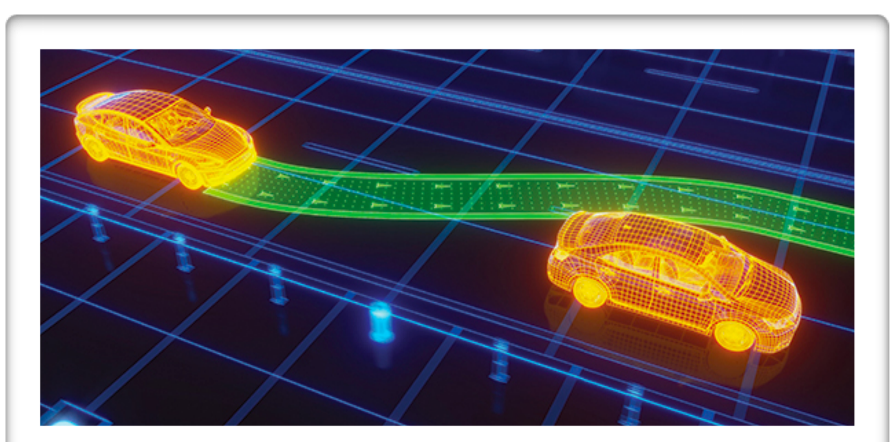



Brave's news recommendation



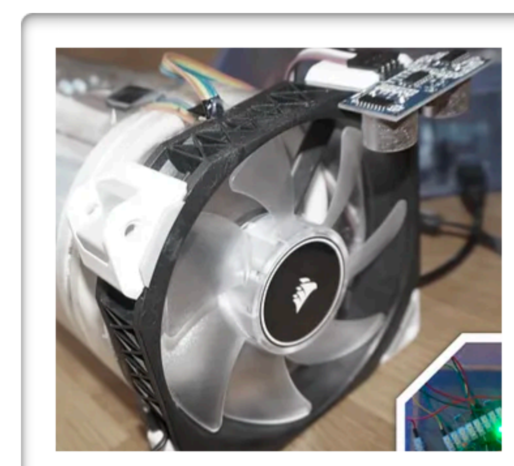

Firefox's URL bar suggestion

IoT



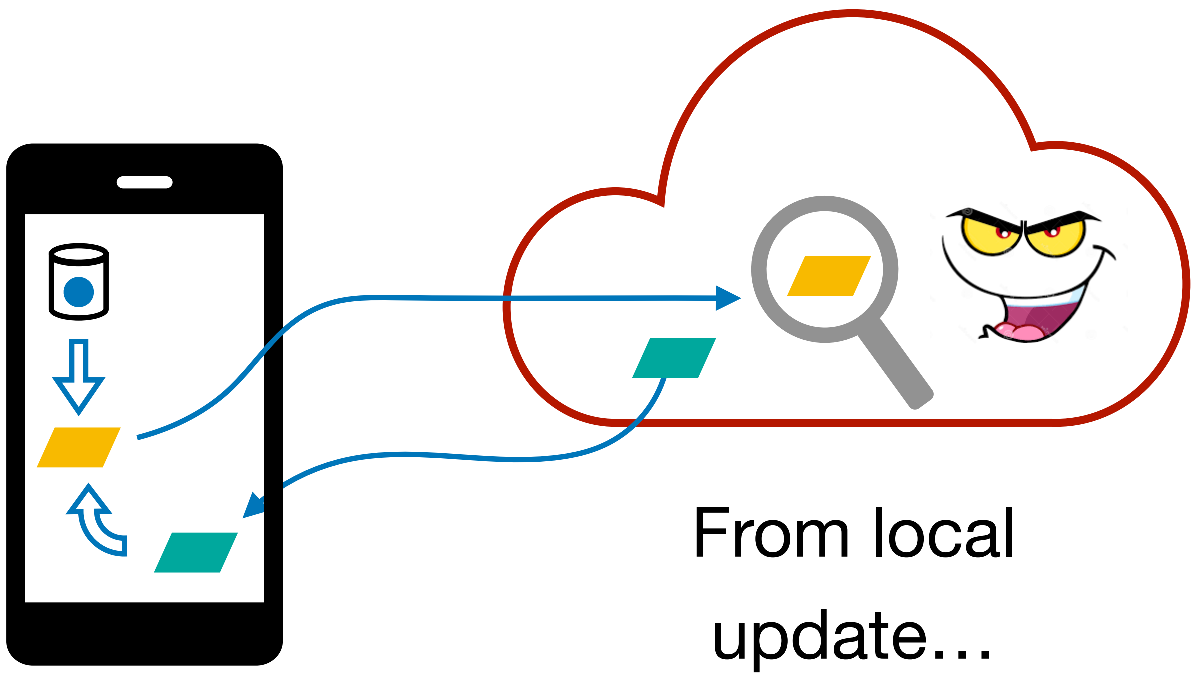
Volvo's trajectory prediction

Cisco's 3D printing

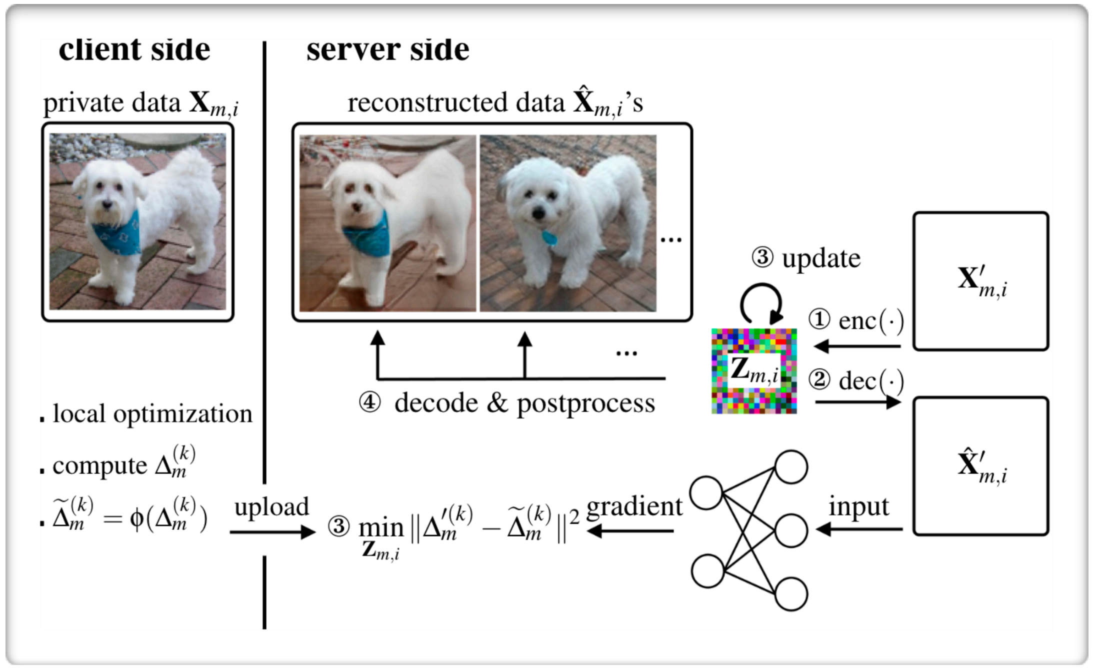


Leveno's clogging detection

# Data Leakage Remains in FL



From local update…




e.g., data reconstruction[1] (Security '23)



[1] Gradient Obfuscation Gives a False Sense of Security in Federated Learning

# Data Leakage Remains in FL

To conceal local updates?

Secure aggregation[1,2]
(CCS ’17, ‘20)

[1] Practical secure aggregation for privacy-preserving machine learning
[2] Secure Single-Server Aggregation with (Poly) Logarithmic Overhead

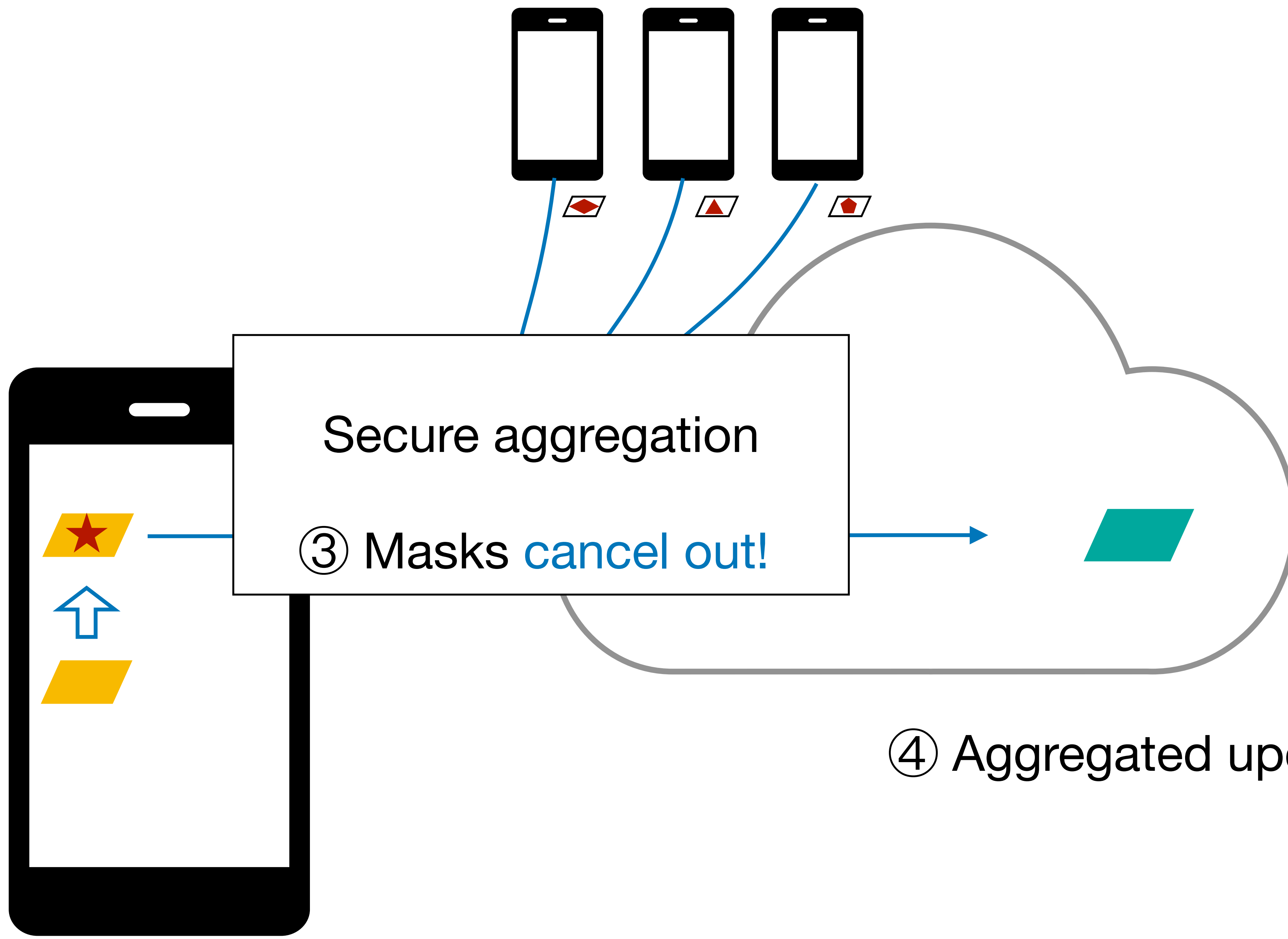
# Data Leakage Remains in FL

To conceal local updates?

② Masked

local update

① Local update




④ Aggregated update

# Data Leakage Remains in FL

To conceal local updates?

To also perturb the aggregated update?

Sacrifice the precision

Differential Privacy[1]

For enhanced privacy



[1] Calibrating Noise to Sensitivity in Privacy Data Analysis

# Data Leakage Remains in FL

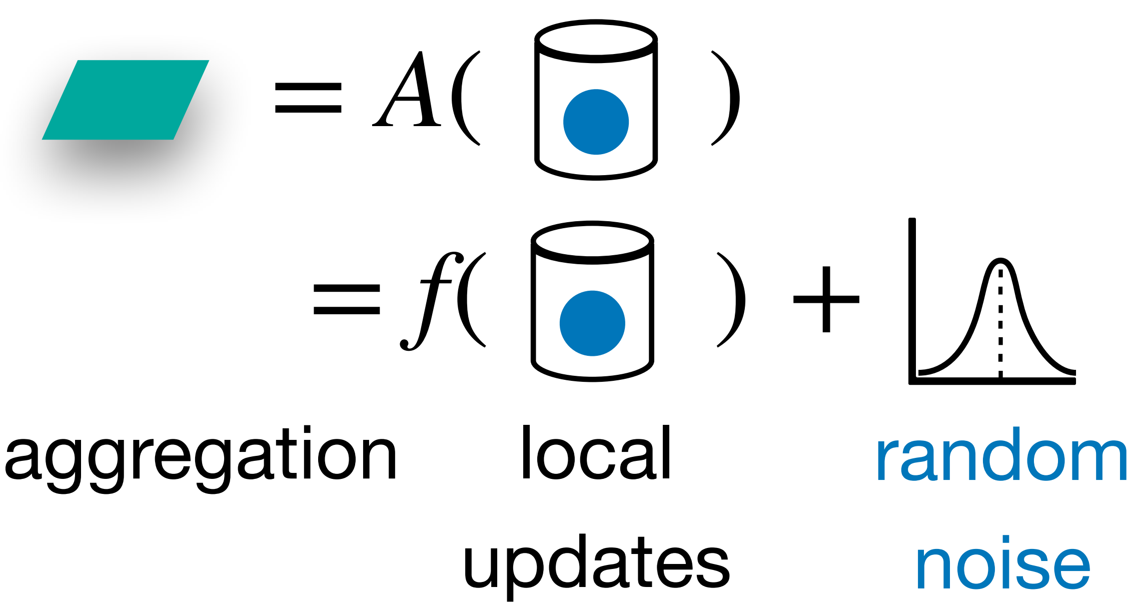
To conceal local updates?

To also perturb the aggregated update?

Sacrifice the precision

Differential Privacy[1]

For enhanced privacy




DP ensures that be **insensitive** to the impact of any single local update in

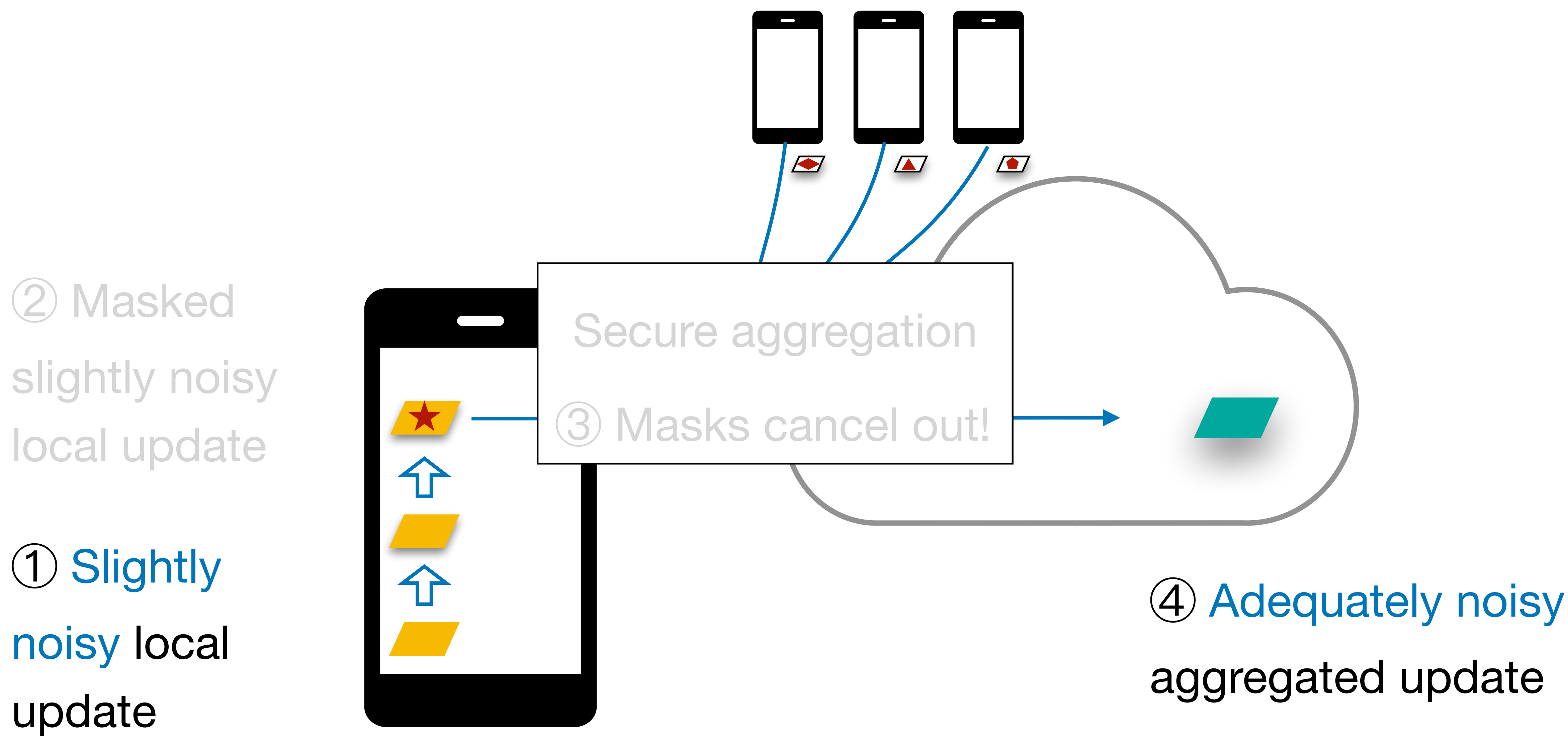


[1] Calibrating Noise to Sensitivity in Privacy Data Analysis

# Data Leakage Remains in FL

To conceal local updates?

To also perturb the aggregated update?



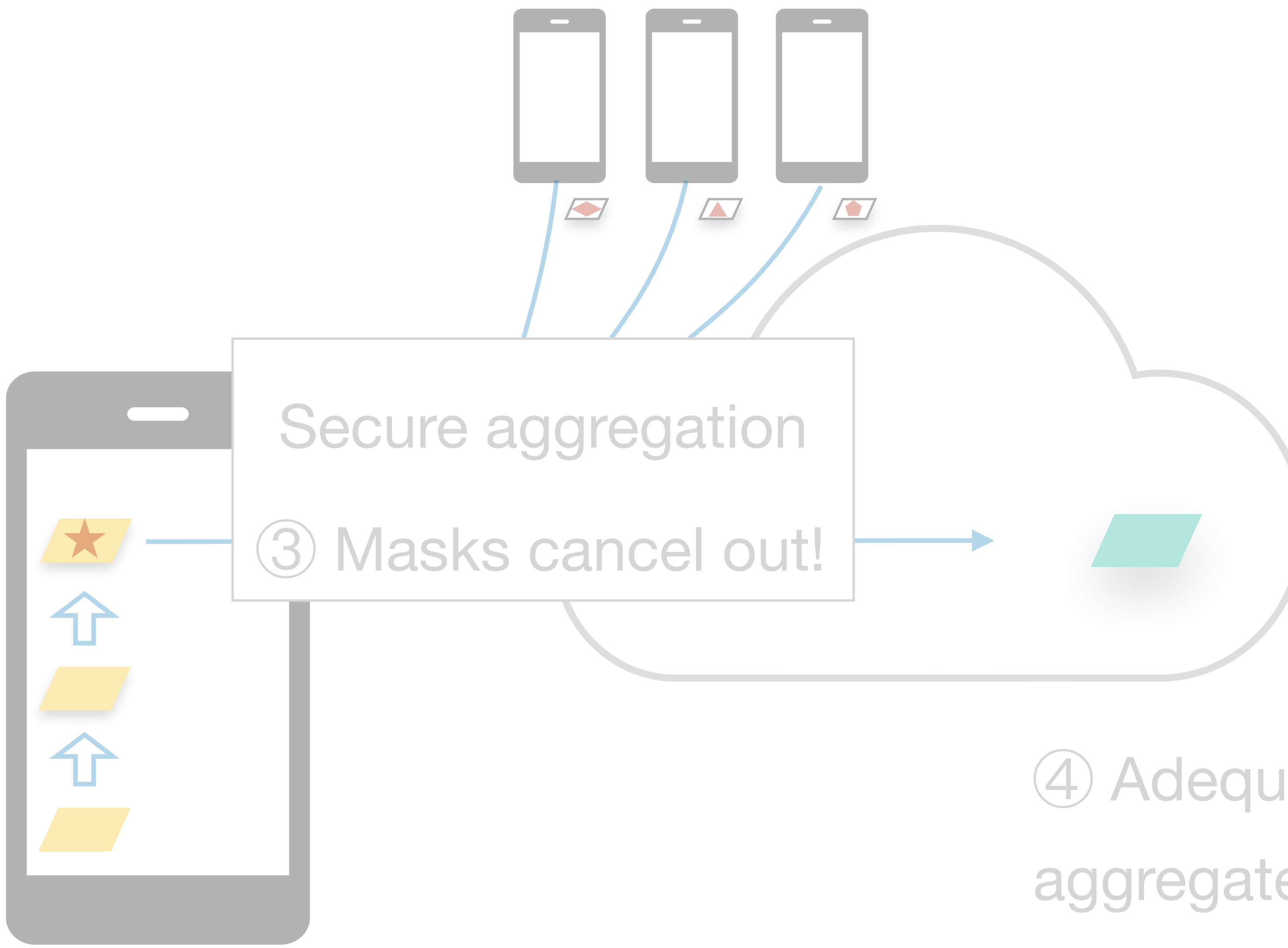

⓪ Global privacy budget $\epsilon \rightarrow$ Calculate the minimum required noise

# Distributed DP = SecAgg + DP

To conceal local updates?
To also perturb the aggregated update?

② Masked slightly noisy local update

① Slightly noisy local update

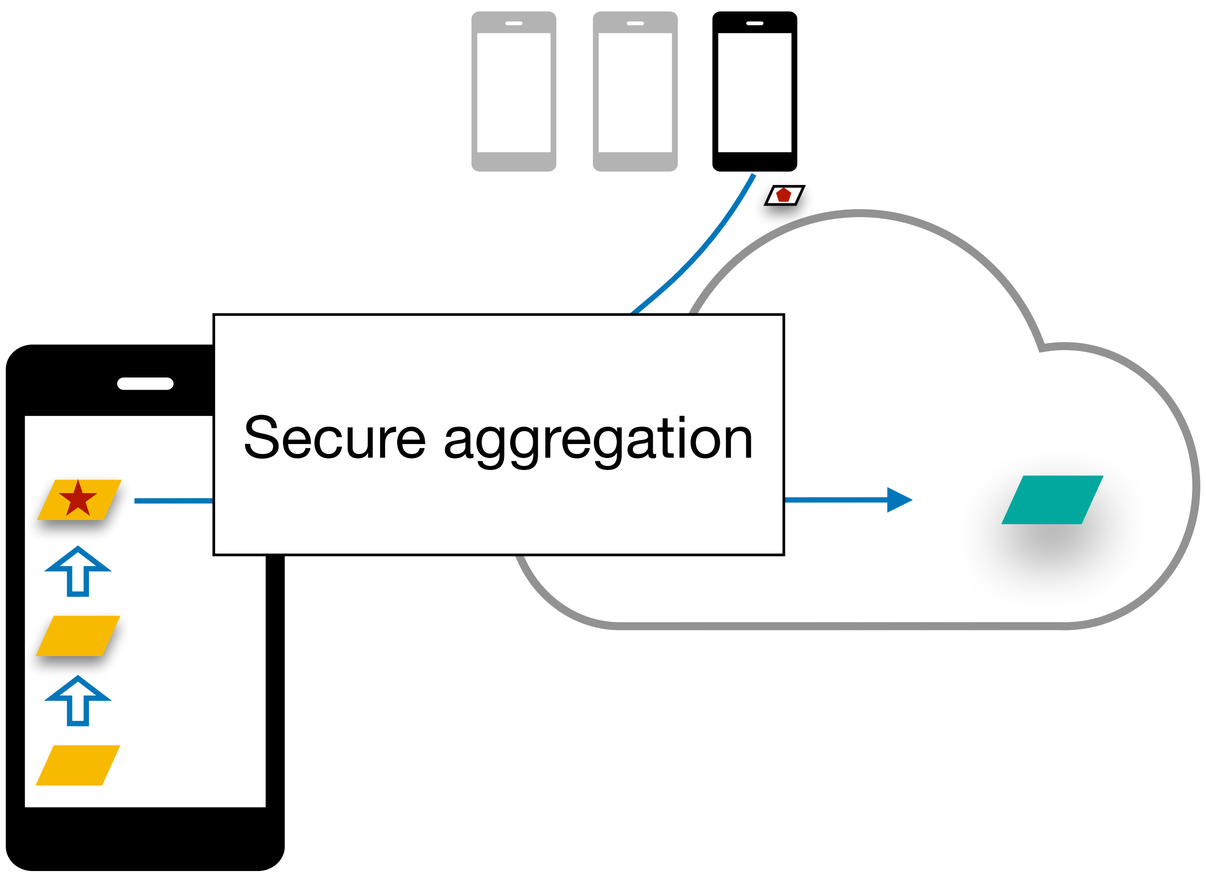



④ Adequately noisy aggregated update

⓪ Global privacy budget $\epsilon \rightarrow$ Calculate the minimum required noise

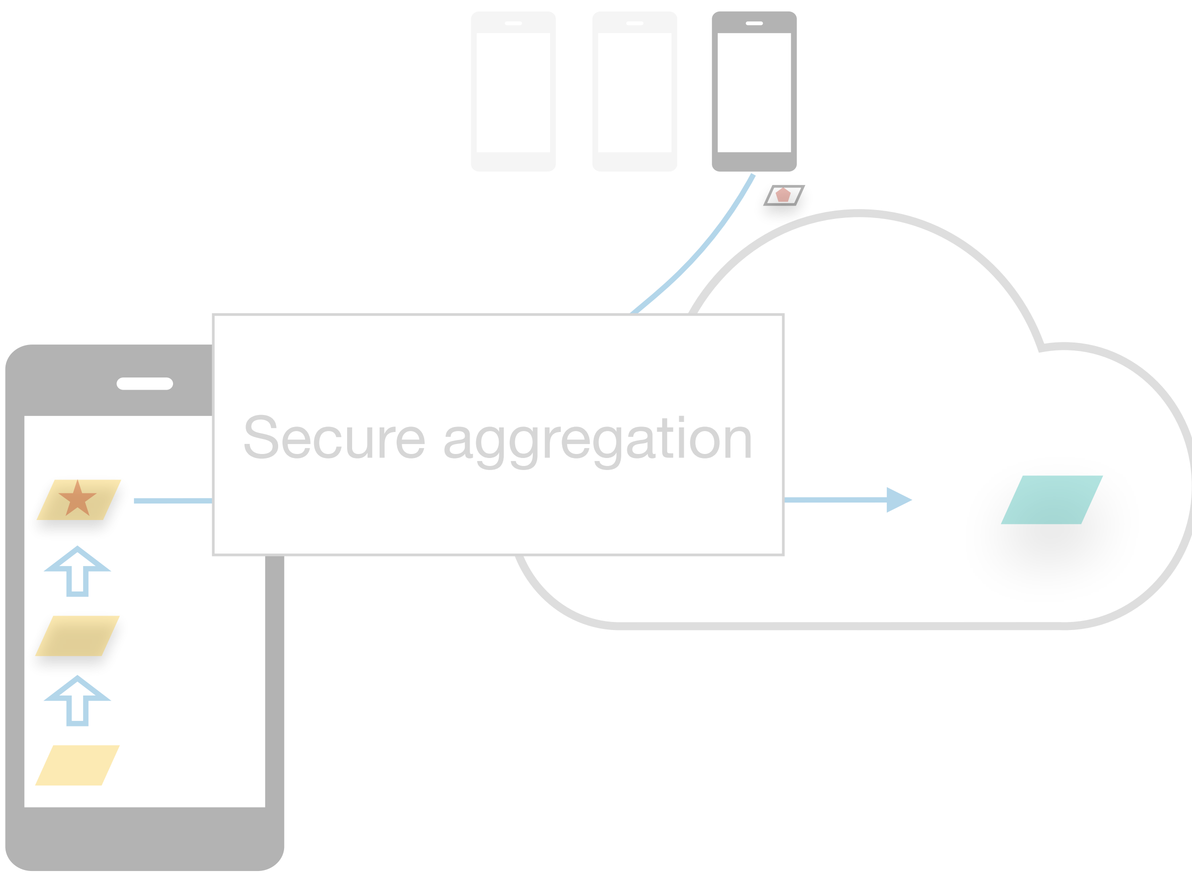
# Distributed DP Has Two Practical Issues

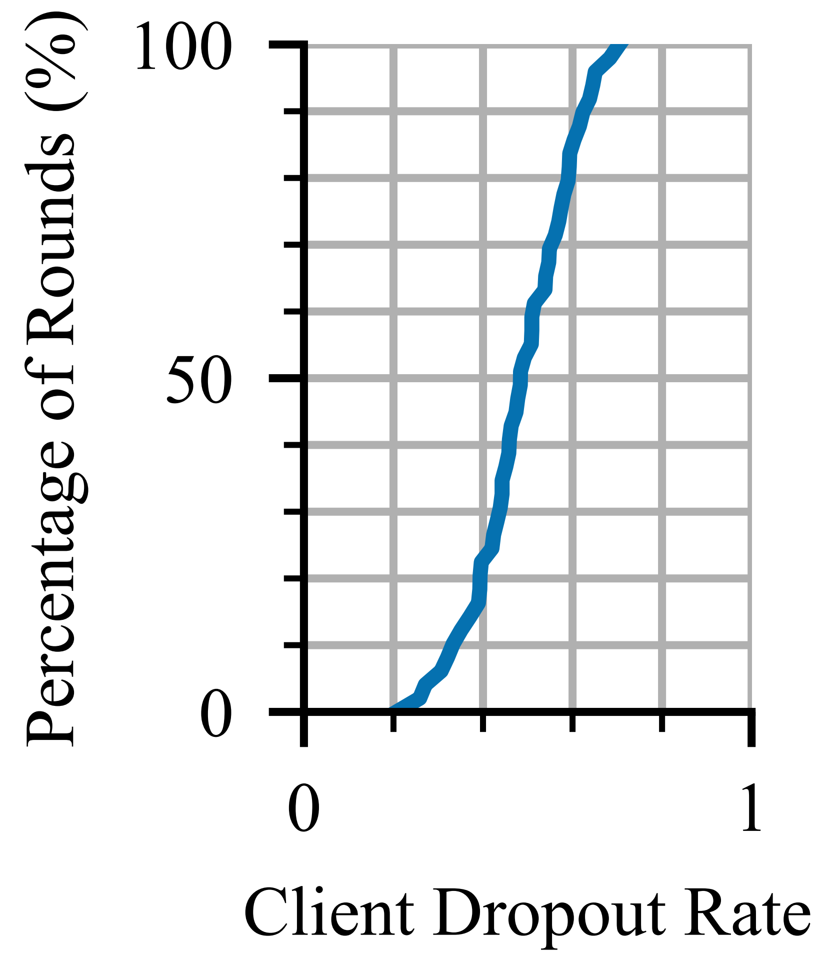
1. **Privacy** Issue: caused by client dropout



# Privacy Issue Caused by Client Dropout

Client dropout can occur anytime



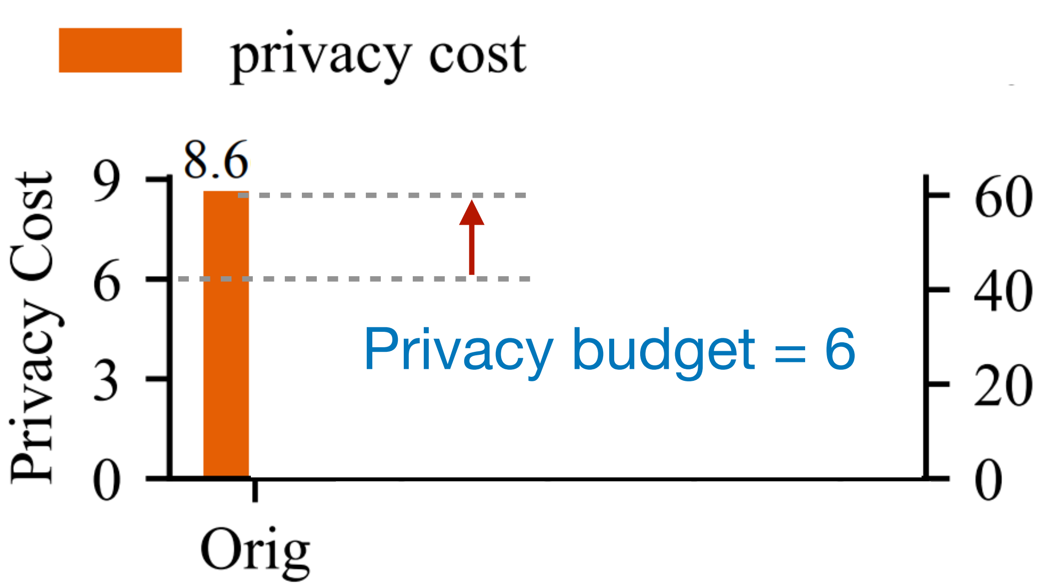

Client behaviors simulated with 100 volatile users from the FLASH dataset[1] (WWW '21)

[1] Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data

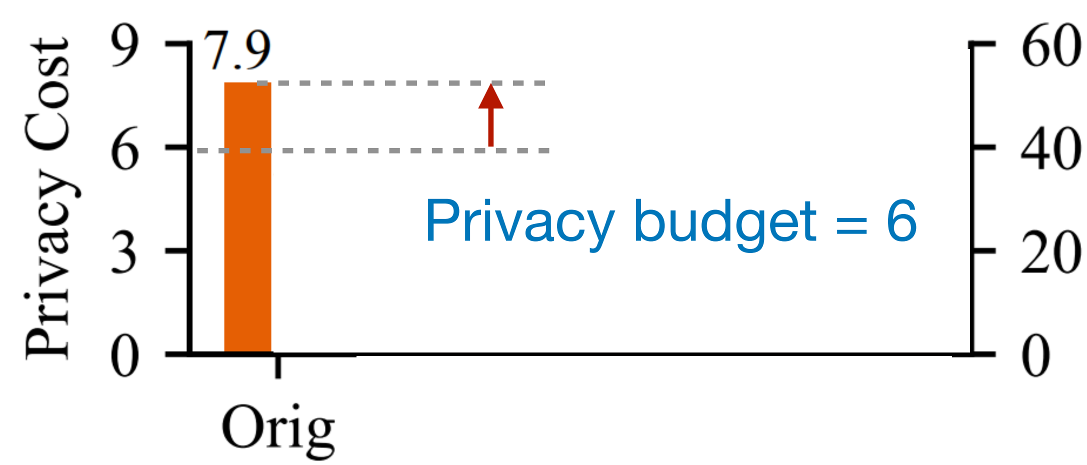
# Privacy Issue Caused by Client Dropout

Client dropout can occur anytime

Insufficient noise for target privacy




CIFAR-10




CIFAR-100

# Dropout-Resilient Noise Enforcement

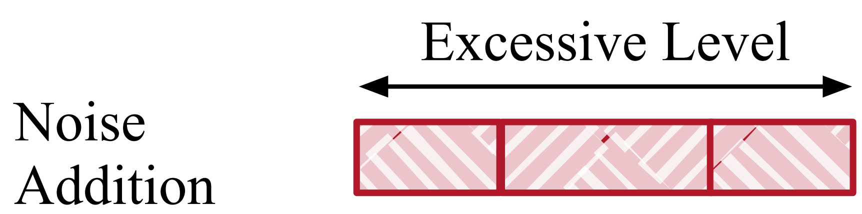
Goal: always enforce the target noise level

# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

- Each client first adds excessive noise as separate components

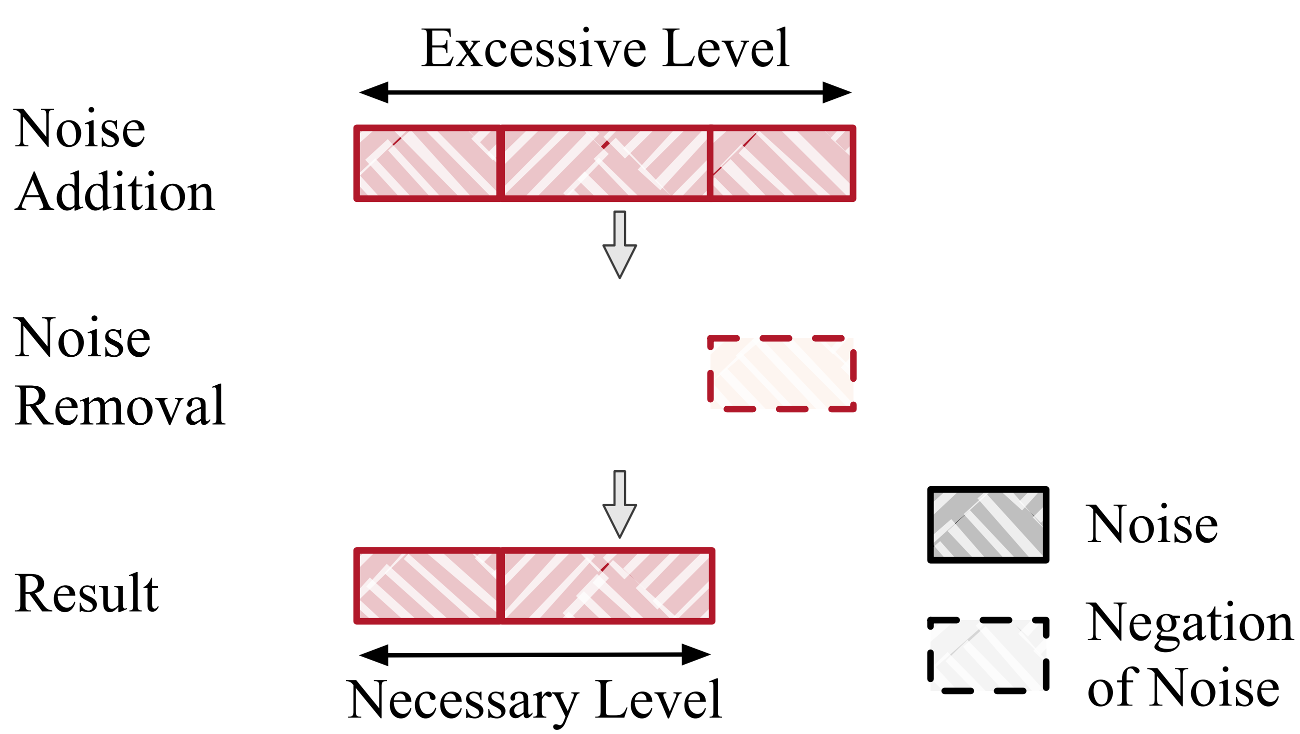


# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

- Each client first adds excessive noise as separate components
- After aggregation, unnecessary ones are removed by the server



# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

Concrete example

Sampled clients $|S| = 4$

Minimum necessary noise level $\sigma_*^2 = 1$

# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

Concrete example

Each client adds noise $n_i \sim \chi(1/2)$
to tolerate up to 2 clients to drop

Add

Sampled clients $|S| = 4$

Dropout tolerance $t = 2,$

Minimum necessary noise level $\sigma_*^2 = 1$

# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

Concrete example

Add

Sampled clients $|S| = 4$

Dropout tolerance $t = 2$,

Minimum necessary noise level $\sigma_*^2 = 1$

Clients

Each client adds noise $n_i \sim \chi(1/2)$
to tolerate up to 2 clients to drop

| | | | |
|---|---|---|---|
| Client 1 | $n_{1,0} \sim \chi(1/4)$ | $n_{1,1} \sim \chi(1/12)$ | $n_{1,2} \sim \chi(1/6)$ |
| Client 2 | $n_{2,0} \sim \chi(1/4)$ | $n_{2,1} \sim \chi(1/12)$ | $n_{2,2} \sim \chi(1/6)$ |
| Client 3 | $n_{3,0} \sim \chi(1/4)$ | $n_{3,1} \sim \chi(1/12)$ | $n_{3,2} \sim \chi(1/6)$ |
| Client 4 | $n_{4,0} \sim \chi(1/4)$ | $n_{4,1} \sim \chi(1/12)$ | $n_{4,2} \sim \chi(1/6)$ |

# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

Concrete example

Add

Sampled clients $|S| = 4$

Dropout tolerance $t = 2,$

Minimum necessary noise level $\sigma_*^2 = 1$

Clients

Each client adds noise $n_i \sim \chi(1/2)$ to tolerate up to 2 clients to drop

$$\begin{array}{lll} n_{1,0} \sim \chi(1/4) & n_{1,1} \sim \chi(1/12) & n_{1,2} \sim \chi(1/6) \\ n_{2,0} \sim \chi(1/4) & n_{2,1} \sim \chi(1/12) & n_{2,2} \sim \chi(1/6) \\ n_{3,0} \sim \chi(1/4) & n_{3,1} \sim \chi(1/12) & n_{3,2} \sim \chi(1/6) \\ n_{4,0} \sim \chi(1/4) & n_{4,1} \sim \chi(1/12) & n_{4,2} \sim \chi(1/6) \end{array}$$

If 0 client drops

Achieve target noise $\sigma_*^2 = 1$

Remove

$$\begin{array}{lll} n_{1,0} & n_{1,1} & n_{1,2} \\ n_{2,0} & n_{2,1} & n_{2,2} \\ n_{3,0} & n_{3,1} & n_{3,2} \\ n_{4,0} & n_{4,1} & n_{4,2} \end{array}$$

To remove

# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

Concrete example

Add

Sampled clients $|S| = 4$

Dropout tolerance $t = 2$,

Minimum necessary noise level $\sigma_*^2 = 1$

Clients

Each client adds noise $n_i \sim \chi(1/2)$
to tolerate up to 2 clients to drop

$n_{1,0} \sim \chi(1/4)$ $n_{1,1} \sim \chi(1/12)$ $n_{1,2} \sim \chi(1/6)$

$n_{2,0} \sim \chi(1/4)$ $n_{2,1} \sim \chi(1/12)$ $n_{2,2} \sim \chi(1/6)$

$n_{3,0} \sim \chi(1/4)$ $n_{3,1} \sim \chi(1/12)$ $n_{3,2} \sim \chi(1/6)$

$n_{4,0} \sim \chi(1/4)$ $n_{4,1} \sim \chi(1/12)$ $n_{4,2} \sim \chi(1/6)$

Remove

If 0 client drops

Achieve target noise $\sigma_*^2 = 1$

$n_{1,0}$ $n_{1,1}$ $n_{1,2}$

$n_{2,0}$ $n_{2,1}$ $n_{2,2}$

$n_{3,0}$ $n_{3,1}$ $n_{3,2}$

$n_{4,0}$ $n_{4,1}$ $n_{4,2}$

To remove

If 1 client drops

Achieve target noise $\sigma_*^2 = 1$

$n_{1,0}$ $n_{1,1}$ $n_{1,2}$

$n_{2,0}$ $n_{2,1}$ $n_{2,2}$

$n_{3,0}$ $n_{3,1}$ $n_{3,2}$

$\cancel{n_{4,0}}$ $\cancel{n_{4,1}}$ $\cancel{n_{4,2}}$

To remove

# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

Concrete example

**Add**

Sampled clients $|S| = 4$

Dropout tolerance $t = 2$,

Minimum necessary noise level $\sigma_*^2 = 1$

Clients

Each client adds noise $n_i \sim \chi(1/2)$
to tolerate up to 2 clients to drop

$n_{1,0} \sim \chi(1/4)$ $n_{1,1} \sim \chi(1/12)$ $n_{1,2} \sim \chi(1/6)$

$n_{2,0} \sim \chi(1/4)$ $n_{2,1} \sim \chi(1/12)$ $n_{2,2} \sim \chi(1/6)$

$n_{3,0} \sim \chi(1/4)$ $n_{3,1} \sim \chi(1/12)$ $n_{3,2} \sim \chi(1/6)$

$n_{4,0} \sim \chi(1/4)$ $n_{4,1} \sim \chi(1/12)$ $n_{4,2} \sim \chi(1/6)$

**Remove**

If 0 client drops

Achieve target noise $\sigma_*^2 = 1$

$n_{1,0}$ $n_{1,1}$ $n_{1,2}$

$n_{2,0}$ $n_{2,1}$ $n_{2,2}$

$n_{3,0}$ $n_{3,1}$ $n_{3,2}$

$n_{4,0}$ $n_{4,1}$ $n_{4,2}$

To remove

If 1 client drops

Achieve target noise $\sigma_*^2 = 1$

$n_{1,0}$ $n_{1,1}$ $n_{1,2}$

$n_{2,0}$ $n_{2,1}$ $n_{2,2}$

$n_{3,0}$ $n_{3,1}$ $n_{3,2}$

$\cancel{n_{4,0}}$ $\cancel{n_{4,1}}$ $\cancel{n_{4,2}}$

To remove

If 2 client drop

Achieve target noise $\sigma_*^2 = 1$

$n_{1,0}$ $n_{1,1}$ $n_{1,2}$

$n_{2,0}$ $n_{2,1}$ $n_{2,2}$

$\cancel{n_{3,0}}$ $\cancel{n_{3,1}}$ $\cancel{n_{3,2}}$

$\cancel{n_{4,0}}$ $\cancel{n_{4,1}}$ $\cancel{n_{4,2}}$

# Dropout-Resilient Noise Enforcement

Goal: always enforce the target noise level

Intuition: add-then-remove

Concrete example

Formal definition: **XNoise**

- Add: decompose $i$'s added noise $n_i \sim \chi\left(\frac{\sigma_*^2}{|S|-t}\right)$ into $t+1$ components:

$$n_i = \sum_{k=0}^{t} n_{i,k},\ n_{i,0} \sim \chi\left(\frac{\sigma_*^2}{|S|}\right), \text{ and } n_{i,k} \sim \chi\left(\frac{\sigma_*^2}{(|S|-k+1)(|S|-k)}\right)(k \in [t])$$

- Remove: when there are $|D|$ clients dropping out, the noise components $n_{i,k}$ contributed by the surviving clients $i \in S \backslash D$ with the index $k > |D|$ becomes excessive and is removed by the server

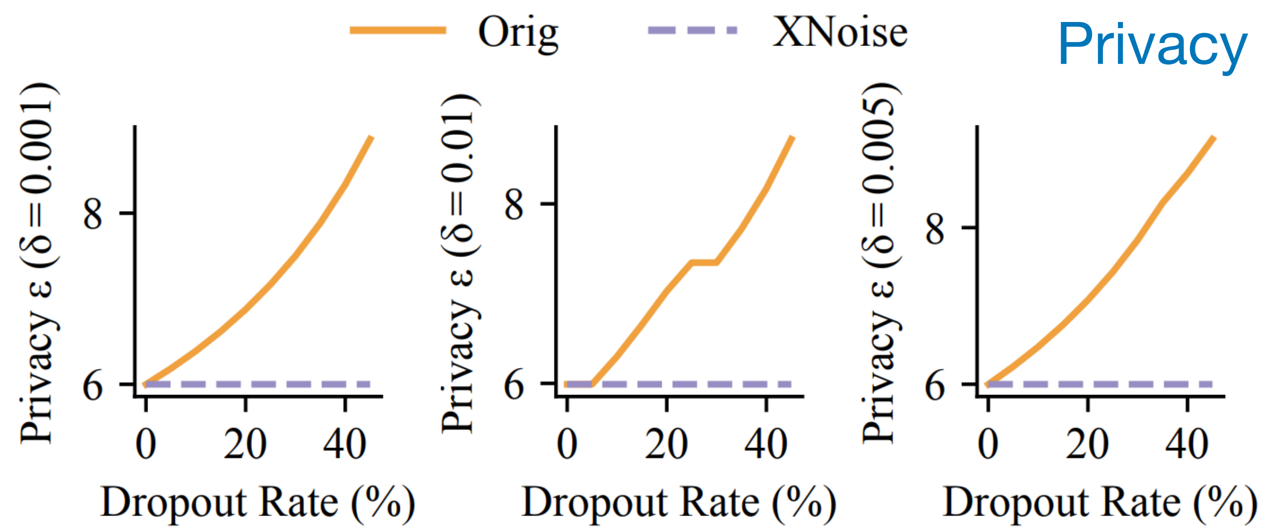
Practical Design:

- Avoiding cascading dropout: secret sharing
- Integrity of the dropout outcome: secure signature

# Dropout-Resilient Noise Enforcement

**XNoise**

Improves privacy




**(a)** FEMNIST. **(b)** CIFAR-10. **(c)** Reddit.

Privacy budget = 6

without sacrificing final model utility

Dropout rates

| *d* | 0 | | 10% | | 20% | | 30% | | 40% | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Ori | XNo | Ori | XNo | Ori | XNo | Ori | XNo | Ori | XNo |
| F | 61.3 | 61.4 | 61.4 | 61.4 | 61.2 | 61.4 | 61.2 | 61.2 | 61.4 | 61.5 |
| C | 66.5 | 66.3 | 66.7 | 66.9 | 66.6 | 65.7 | 64.3 | 65.7 | 63.8 | 64.2 |
| R | 2169 | 2142 | 2158 | 2179 | 2286 | 2285 | 2294 | 2317 | 2299 | 2329 |

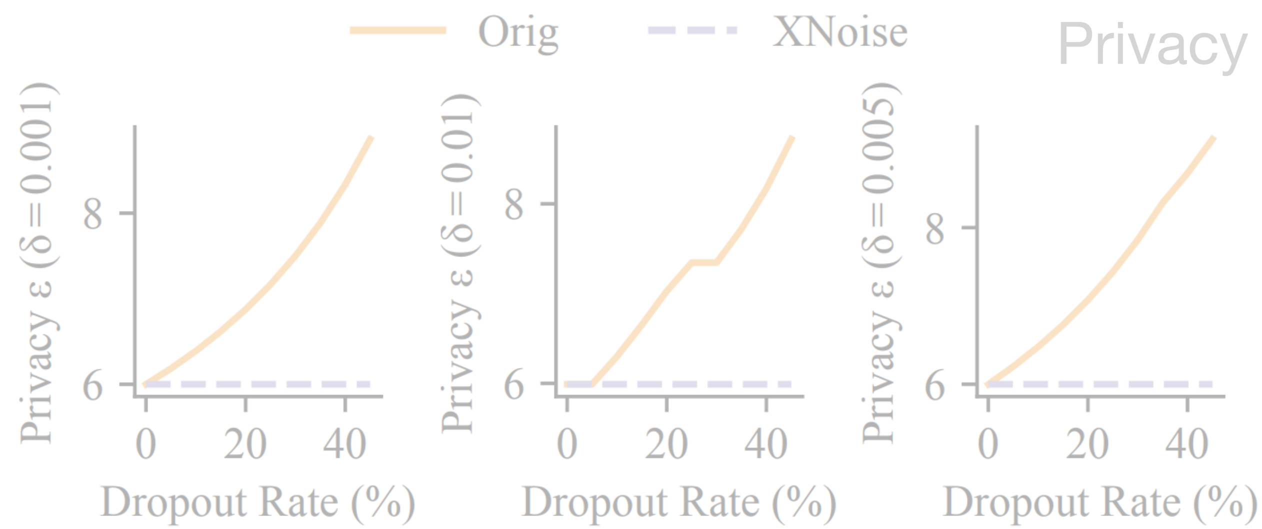
Datasets

# Dropout-Resilient Noise Enforcement

**XNoise**

Improves privacy

Privacy budget = 6



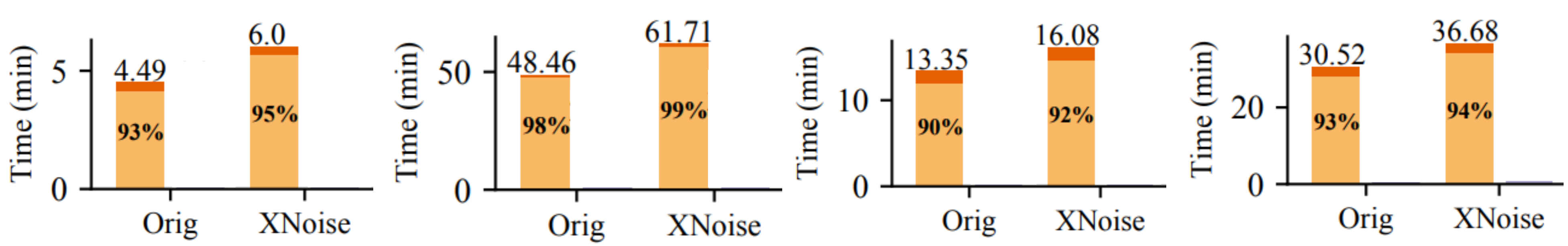

(a) FEMNIST. (b) CIFAR-10. (c) Reddit.

without sacrificing final model utility

Dropout rates

| Datasets / d | 0 | | 10% | | 20% | | 30% | | 40% | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Ori | XNo | Ori | XNo | Ori | XNo | Ori | XNo | Ori | XNo |
| F | 61.3 | 61.4 | 61.4 | 61.4 | 61.2 | 61.4 | 61.2 | 61.2 | 61.4 | 61.5 |
| C | 66.5 | 66.3 | 66.7 | 66.9 | 66.6 | 65.7 | 64.3 | 65.7 | 63.8 | 64.2 |
| R | 2169 | 2142 | 2158 | 2179 | 2286 | 2285 | 2294 | 2317 | 2299 | 2329 |

and incurs acceptable ( $\leq$ 34 % ) runtime cost




FEMNIST@CNN FEMNIST@ResNet18 CIFAR10@ResNet18 CIFAR10@VGG19

Example: no dropout

# Distributed DP has Two Issues

1. **Privacy** Issue: caused by client dropout
2. **Performance** Issue: expensive use of secure aggregation

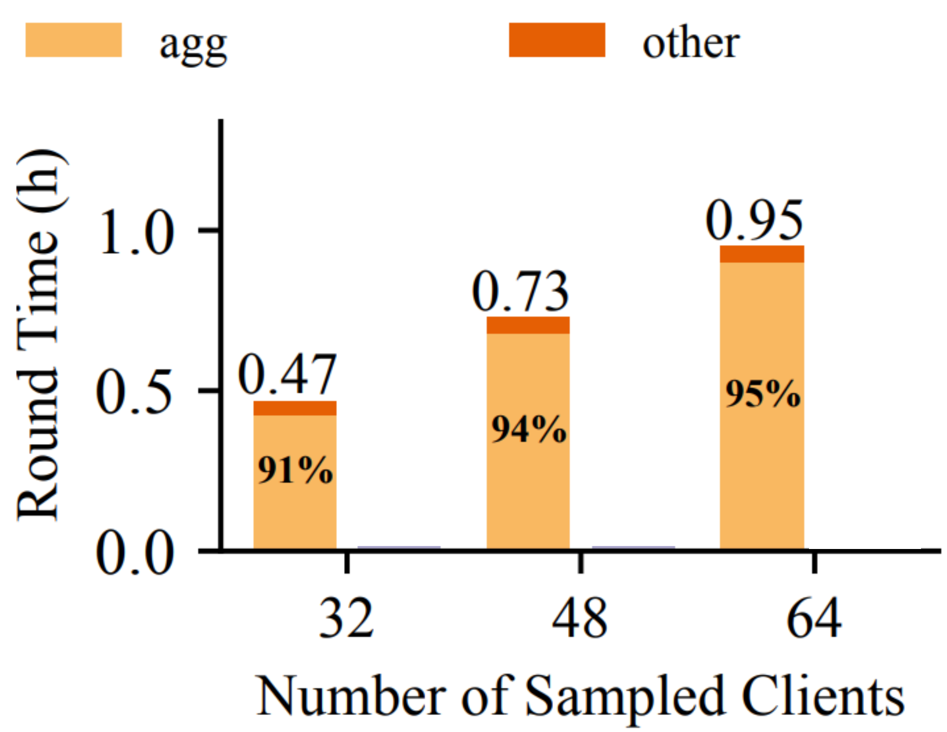
# Performance issues with SecAgg

Extensive use of secret sharing and pairwise masking

# Performance issues with SecAgg

Extensive use of secret sharing and pairwise masking

Dominates the training time (at least 91%)




original secure aggregation:
SecAgg

# Performance issues with SecAgg

Extensive use of secret sharing and pairwise masking

Dominates the training time (at least 91%)

Follow-up solutions

- e.g. SecAgg+: improves asymptotically


original secure aggregation: SecAgg

# Performance issues with SecAgg

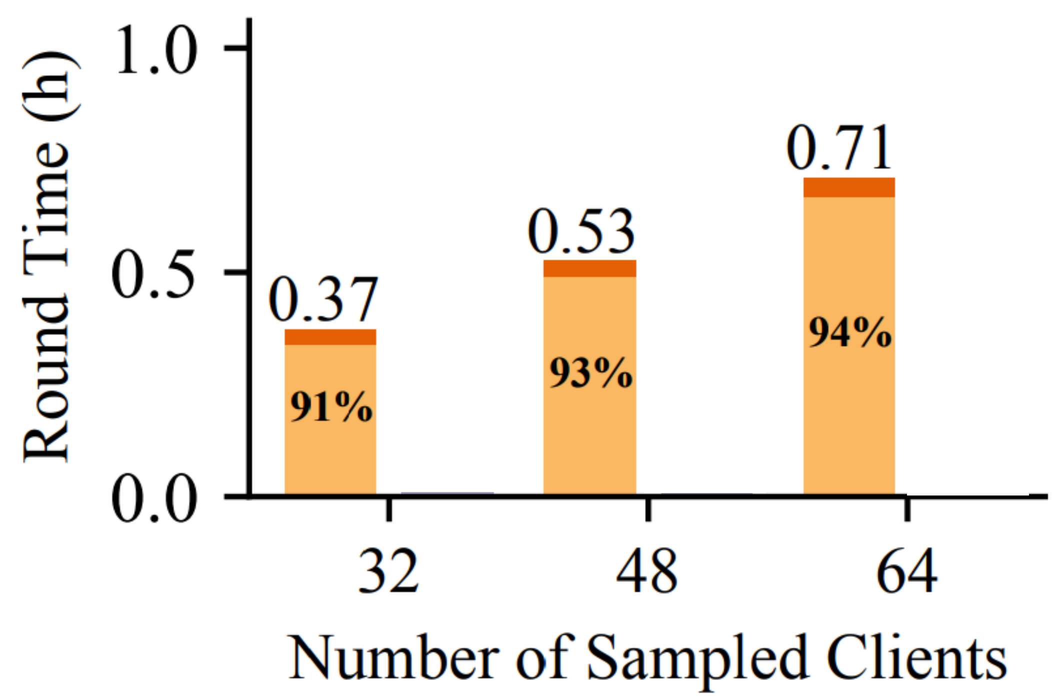
Extensive use of secret sharing and pairwise masking

Dominates the training time (at least 91%)

Follow-up solutions have inefficiencies

- e.g. SecAgg+: improves asymptotically, but help little in small-scale practice[1]


original secure aggregation: SecAgg




SOTA secure aggregation: SecAgg+

[1] Towards federated learning at scale: system design, MLSys ‘19

# Pipeline-Parallel Acceleration

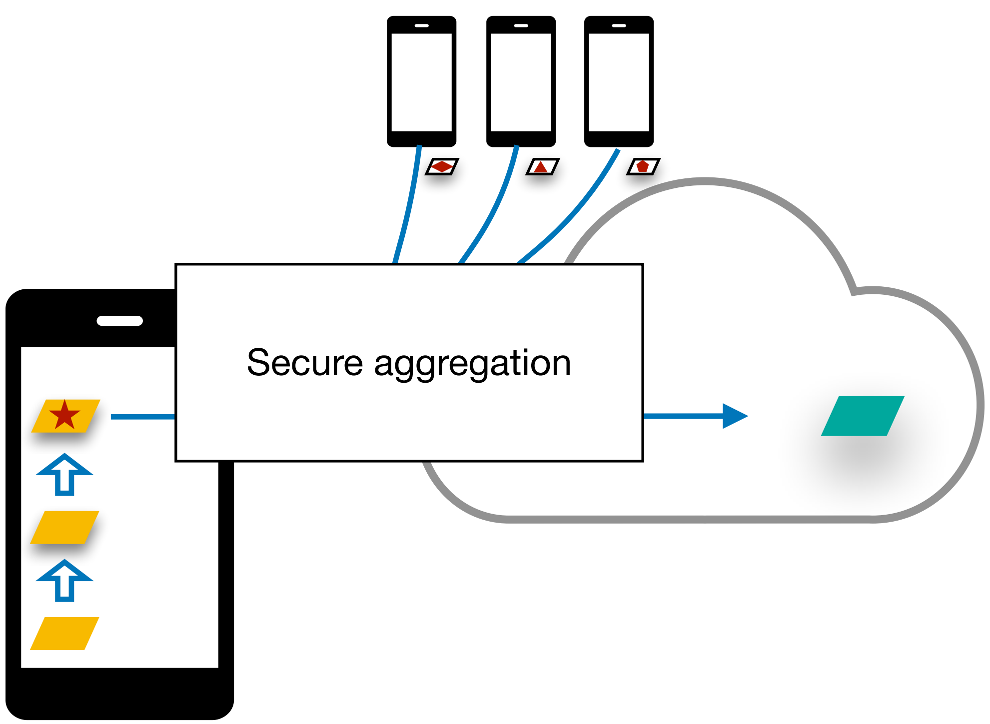
Goal: leverage the underutilized resources in the system level

# Pipeline-Parallel Acceleration

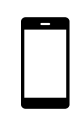
Goal: leverage the underutilized resources in the system level

Approach:

- Step 1: Identify the types of system resources




**s-comp**: *the compute resources (e.g., CPU, GPU, and memory) of the server*



**c-comp**: *the compute resources of clients*

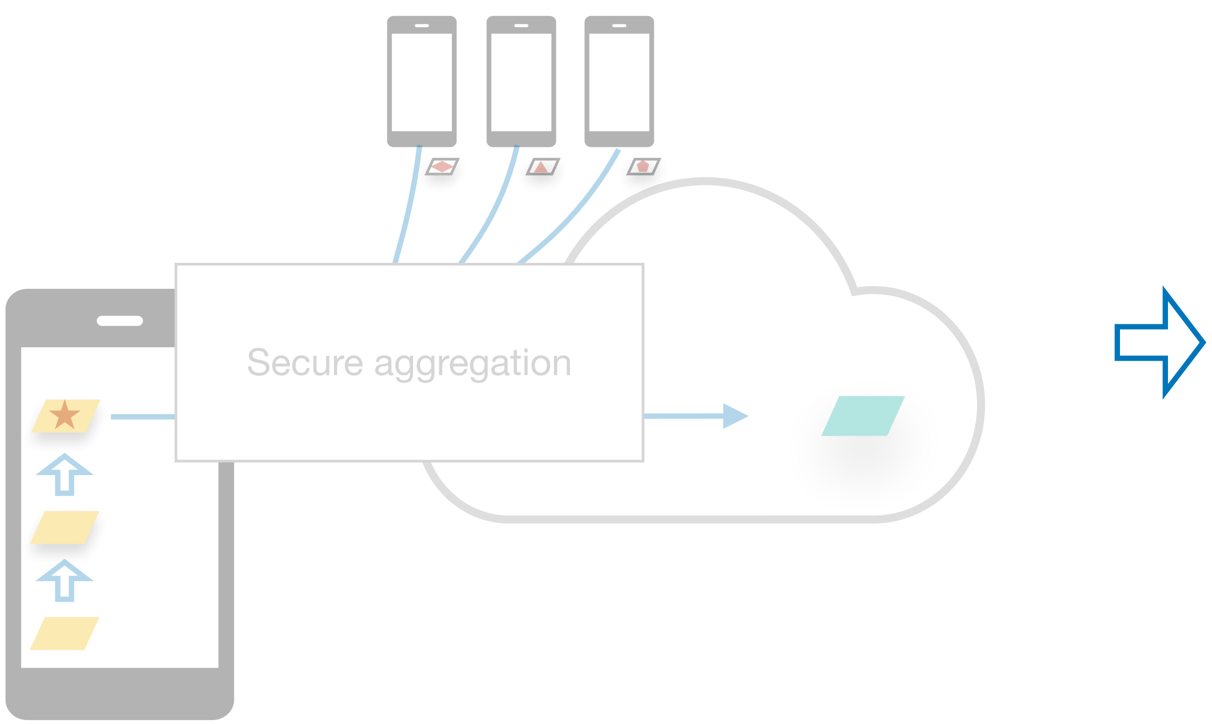
**comm**: *the network resource used for server-client communication*

# Pipeline-Parallel Acceleration

Goal: leverage the underutilized resources in the system level

Approach:

- Step 1: Identify the types of system resources
- Step 2: Group consecutive operations that use the same system resources



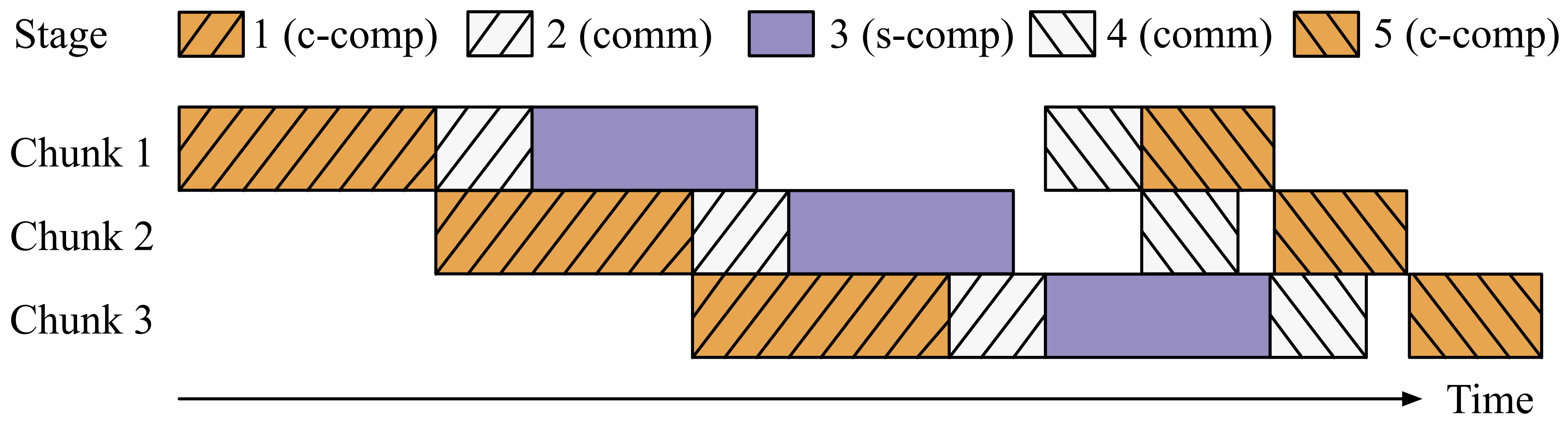

| Step | Operation | Stage (Resource) |
|---|---|---|
| 1 | Clients encode updates. | 1 (c-comp) |
| 2 | Clients generate security keys. | |
| 3 | Clients establish shared secrets. | |
| 4 | Clients mask encoded updates. | |
| 5 | Clients upload masked updates. | 2 (comm) |
| 6 | Server deals with dropout. | 3 (s-comp) |
| 7 | Server computes aggregate update. | |
| 8 | Server updates the global model. | |
| 9 | Server dispatches the aggregate. | 4 (comm) |
| 10 | Clients decode the aggregate. | 5 (c-comp) |
| 11 | Clients use the aggregate. | |

# Pipeline-Parallel Acceleration

Goal: leverage the underutilized resources in the system level

Approach:

- Step 1: Identify the types of system resources
- Step 2: Group consecutive operations that use the same system resources
- Step 3: Evenly partition each client's update into chunks and pipeline them



# Pipeline-Parallel Acceleration

Goal: leverage the underutilized resources in the system level

Approach:

- Step 1: Identify the types of system resources
- Step 2: Group consecutive operations that use the same system resources
- Step 3: Evenly partition each client's update into chunks and pipeline them
    - Optimize to determine the optimal number of chunk, $m^*$

$$
\begin{aligned}
& m^* = \arg\min_{m \in N_+} f_{a,m} \\
s.t. \quad & f_{s,c} = b_{s,c} + l_s \\
& b_{s,c} = \max\{o_{s,c}, r_{s,c}\} \\
& o_{s,c} = \begin{cases} 0, & \text{if } s = 0, \\ f_{s-1,c} \end{cases}
\end{aligned}
$$

*Definition of the finish time of Chunk $m$ at Stage $a$*

*Intra-chunk sequential execution*

$$
r_{s,c} = \begin{cases} 0, & \text{if } s = 0 \text{ and } c = 0, \\ f_{q,m} \text{ or } \perp, & \text{if } s \neq 0 \text{ and } c = 0, \\ f_{s,c-1}, & \text{otherwise} \end{cases}
$$

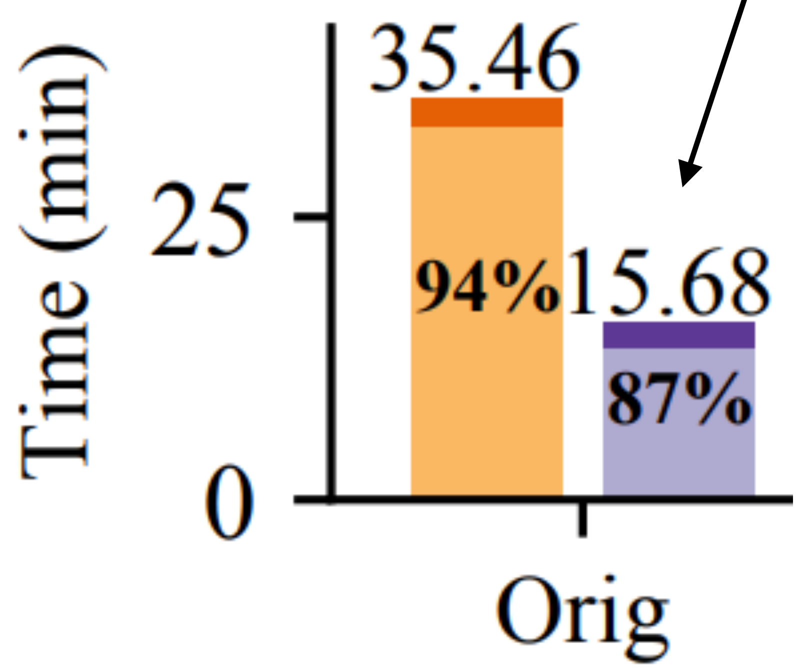
*Exclusive allocation*
*& Inter-chunk sequential execution*

# Pipeline-Parallel Acceleration

Effectiveness:

① A maximum speedup of 2.4×

w/ or w/o pipelining

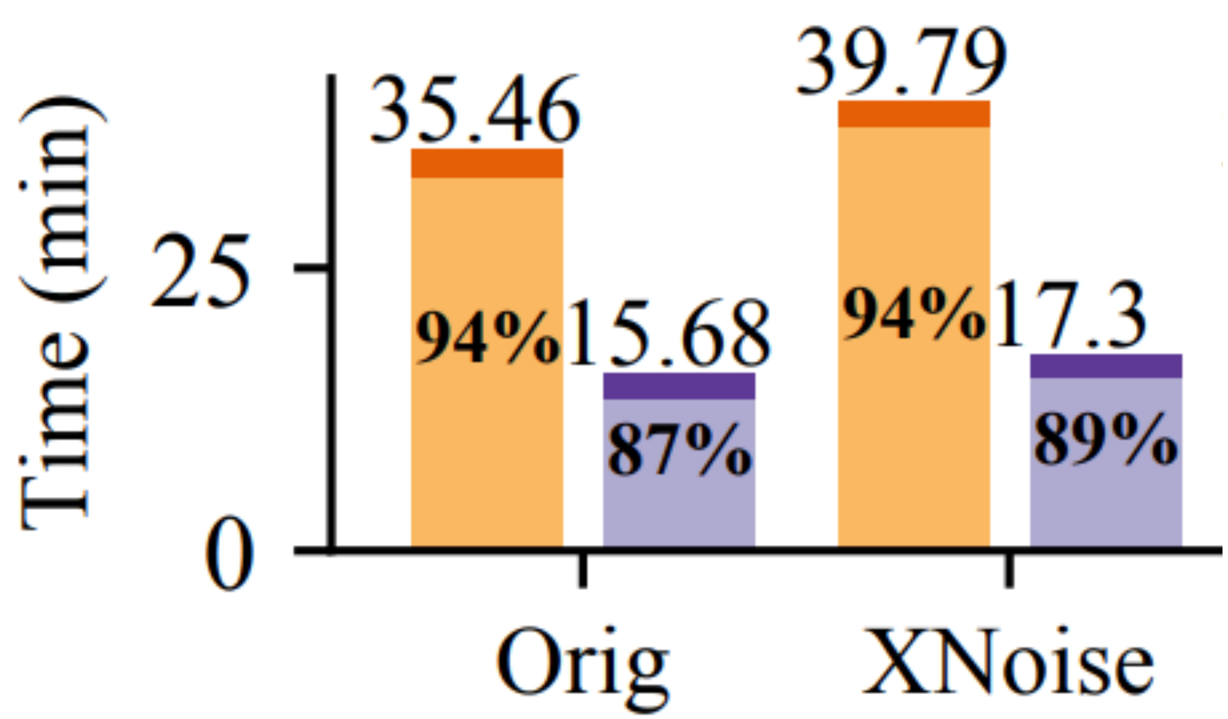



Case study: CIFAR10 @ VGG19, dropout rate = 30%

# Pipeline-Parallel Acceleration

Effectiveness:

① A maximum speedup of 2.4×

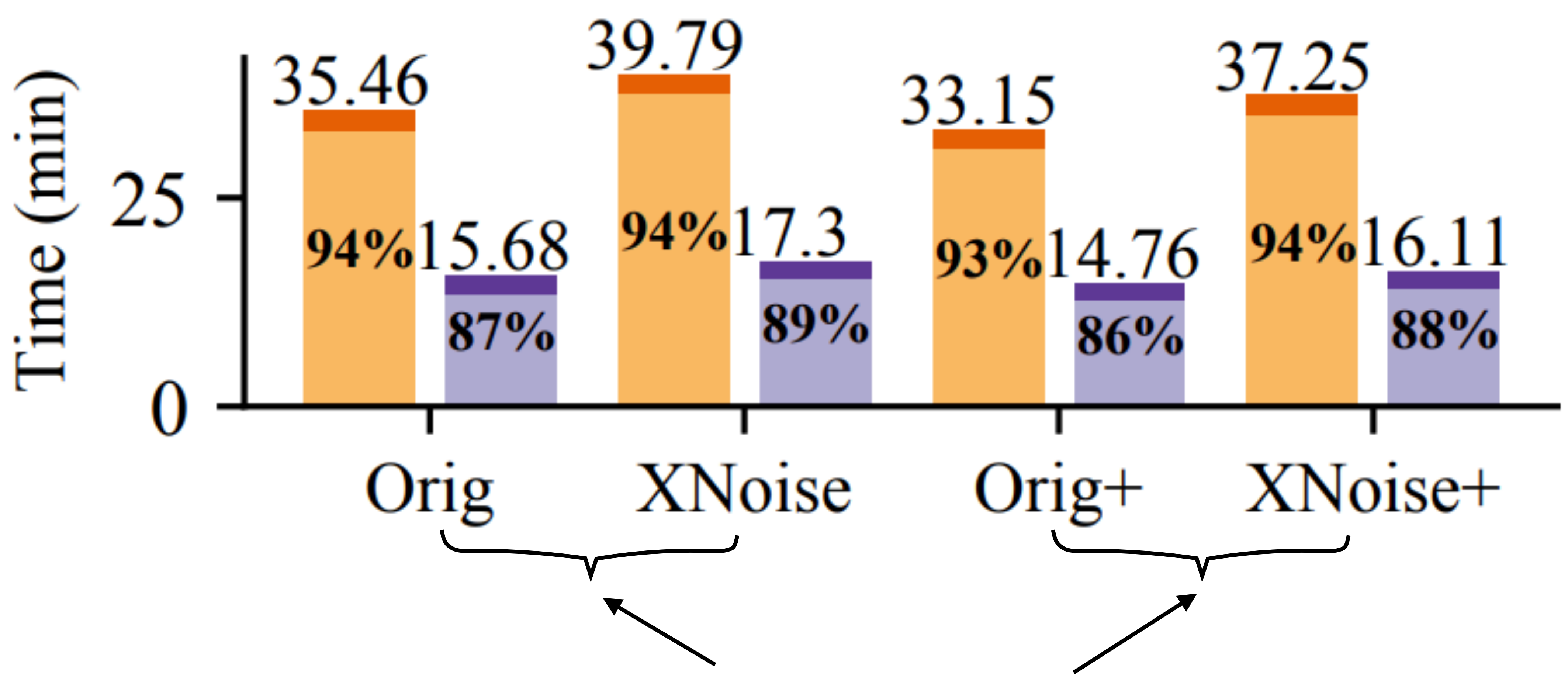



w/o or w/
our noise enforcement

# Pipeline-Parallel Acceleration

Effectiveness:

① A maximum speedup of 2.4×

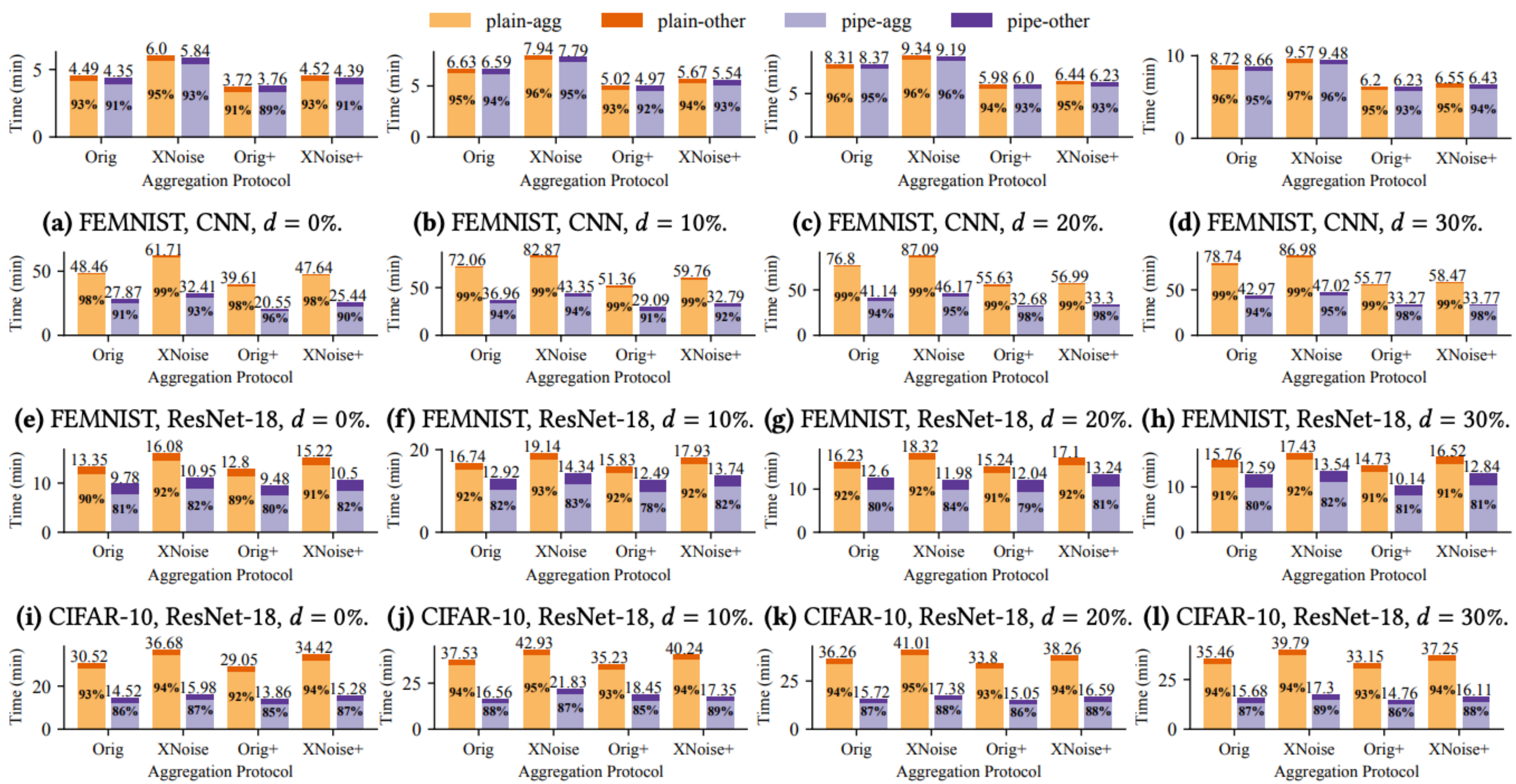



Implemented using
SecAgg or SecAgg+

# Pipeline-Parallel Acceleration

Effectiveness:

① A maximum speedup of 2.4×




(a) FEMNIST, CNN, $d$ = 0%. (b) FEMNIST, CNN, $d$ = 10%. (c) FEMNIST, CNN, $d$ = 20%. (d) FEMNIST, CNN, $d$ = 30%.

(e) FEMNIST, ResNet-18, $d$ = 0%. (f) FEMNIST, ResNet-18, $d$ = 10%. (g) FEMNIST, ResNet-18, $d$ = 20%. (h) FEMNIST, ResNet-18, $d$ = 30%.

(i) CIFAR-10, ResNet-18, $d$ = 0%. (j) CIFAR-10, ResNet-18, $d$ = 10%. (k) CIFAR-10, ResNet-18, $d$ = 20%. (l) CIFAR-10, ResNet-18, $d$ = 30%.

(m) CIFAR-10, VGG-19, $d$ = 0%. (n) CIFAR-10, VGG-19, $d$ = 10%. (o) CIFAR-10, VGG-19, $d$ = 20%. (p) CIFAR-10, VGG-19, $d$ = 30%.

②
Larger
models
gain
more

③
Scaling w/
# part.

④ Gains are consistent across different dropout rates

# Dordis

https://github.com/SamuelGong/Dordis

A distributed DP framework for

- Privacy
- Efficiency

in FL training

Automate

| | | | | |
|---|---|---|---|---|
| Precise Noise Enforcement | For | Privacy Preservation | Against | Client Dropout |
| Optimal Pipelined Execution | For | Efficiency Enhancement | Against | SecAgg's Bottleneck |

## Thank you!

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

NOKIA Bell Labs