上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

Frontier Research of Federated Learning and Serverless Computing

马汝辉 副教授

计算机学院

上海交通大学

Section 1: Frontier Research of Federated Learning

Guarantee Data Confidentiality

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录

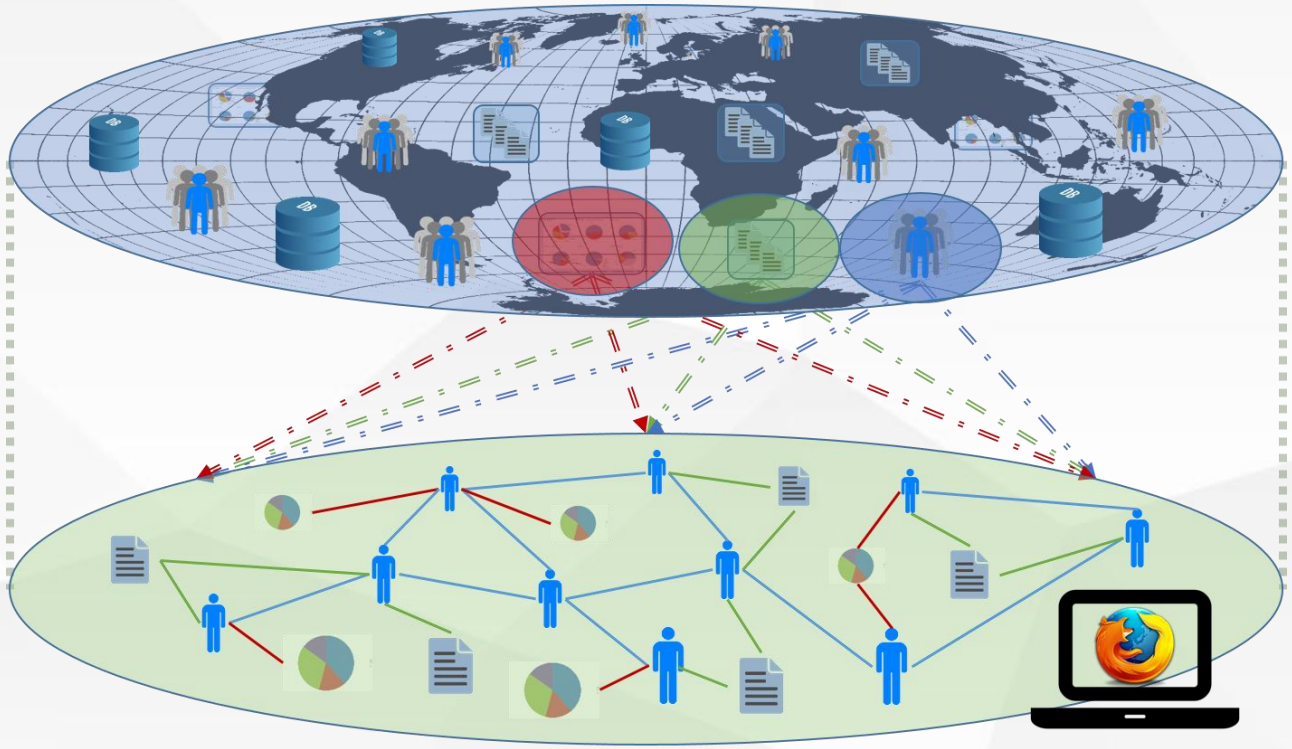
01

Challenges in FL

- Expensive communication
- Systems heterogeneity
- Statistical heterogeneity
- Privacy concerns

Challenges in FL

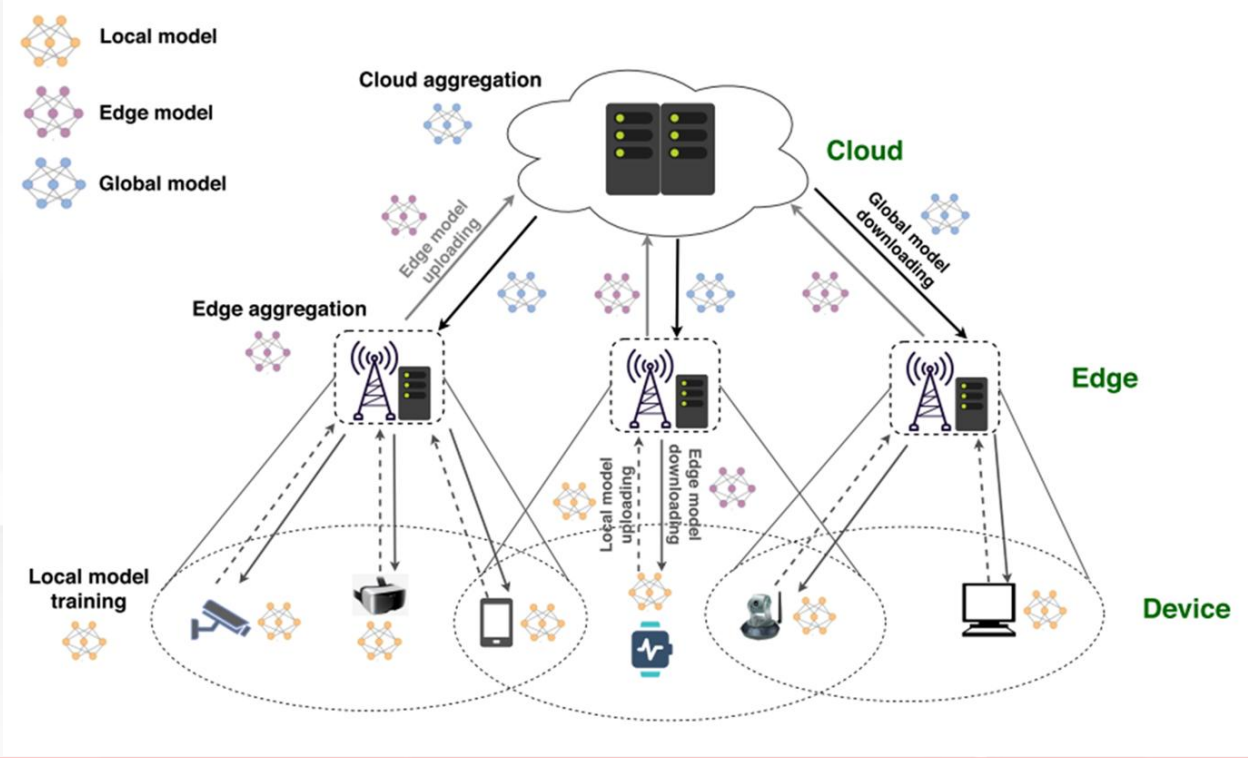
- **Expensive Communication.**
  - federated networks are potentially comprised of a massive number of devices, e.g., millions of smart phones, and communication in the network can be slower than local computation by many orders of magnitude.



Challenges in FL

Systems Heterogeneity.

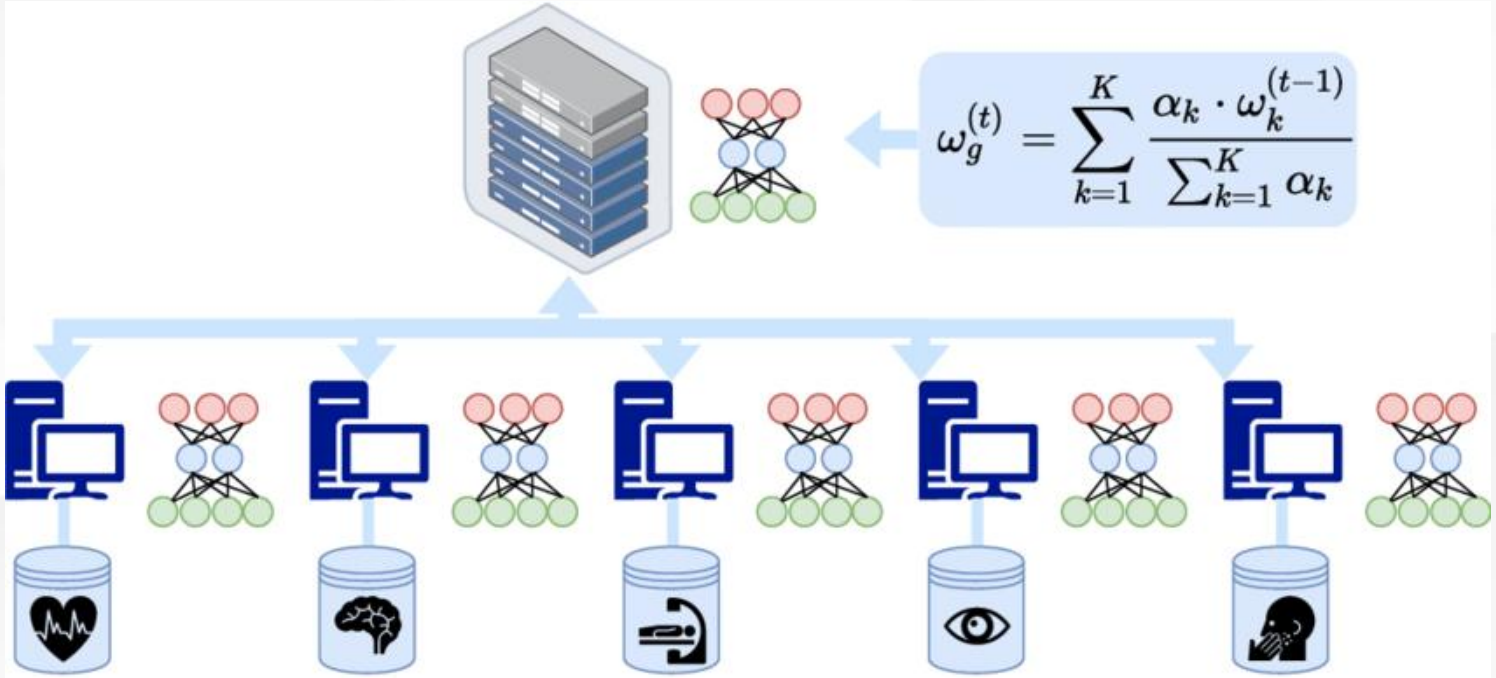
- The storage, computational, and communication capabilities of each device in federated networks may differ due to variability in hardware (CPU, memory), network connectivity (3G, 4G, 5G, wifi), and power (battery level).
- Each device may also be unreliable.



Challenges in FL

Statistical Heterogeneity.

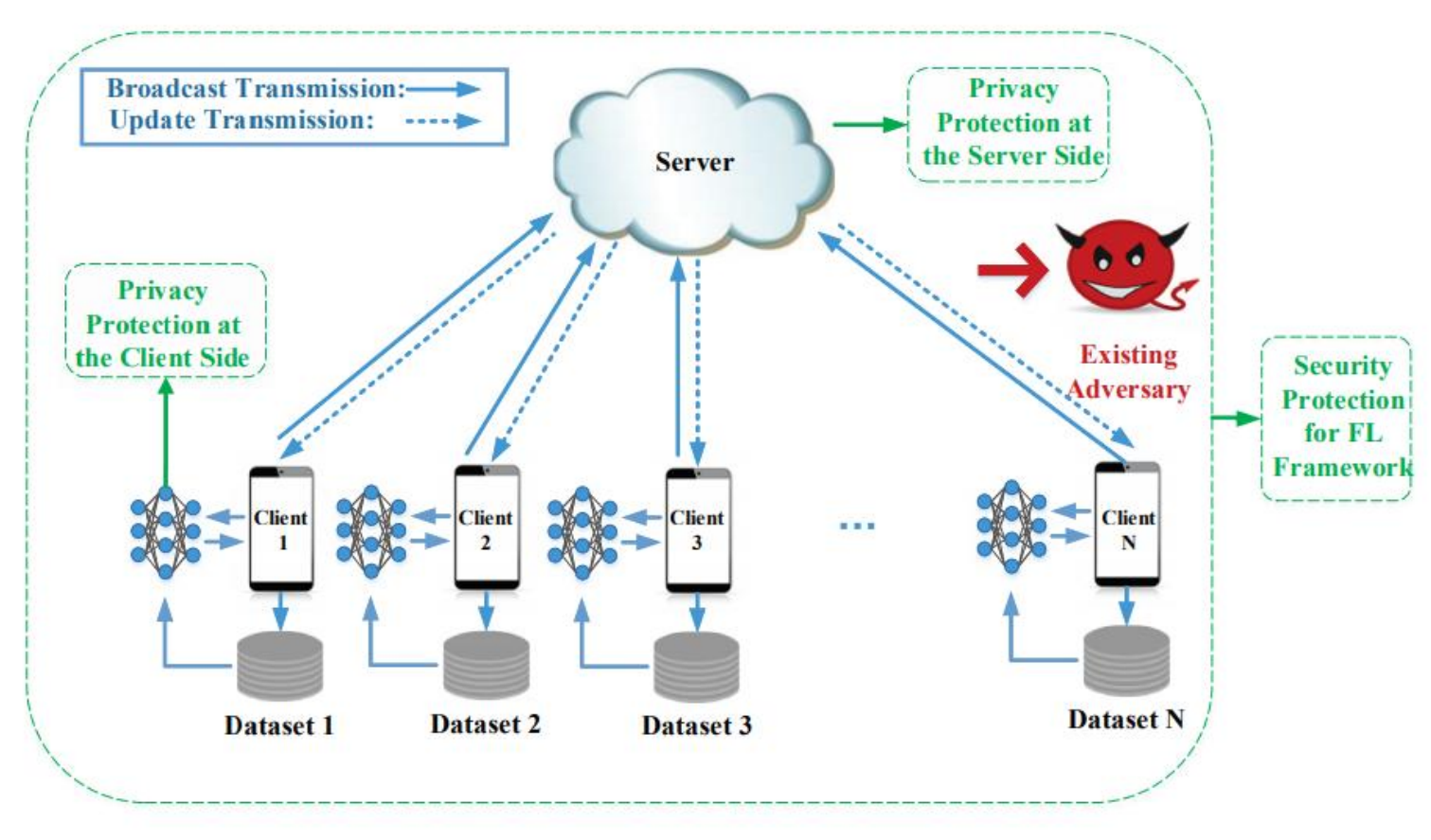
- Devices frequently generate and collect data in a non-identically distributed manner across the network, e.g., mobile phone users have varied use of language in the context of a next word prediction task.
- Increases the likelihood of stragglers.



Challenges in FL

Privacy Concerns.

- communicating model updates throughout the training process can nonetheless reveal sensitive information



02

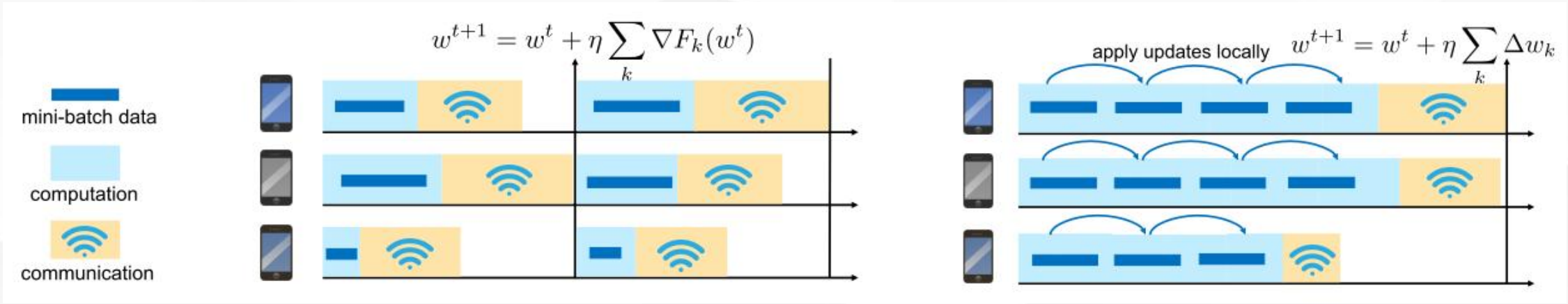
Communication-efficiency

- Local updating
- Compression schemes
- Decentralized training

Communication-efficiency

Local updating

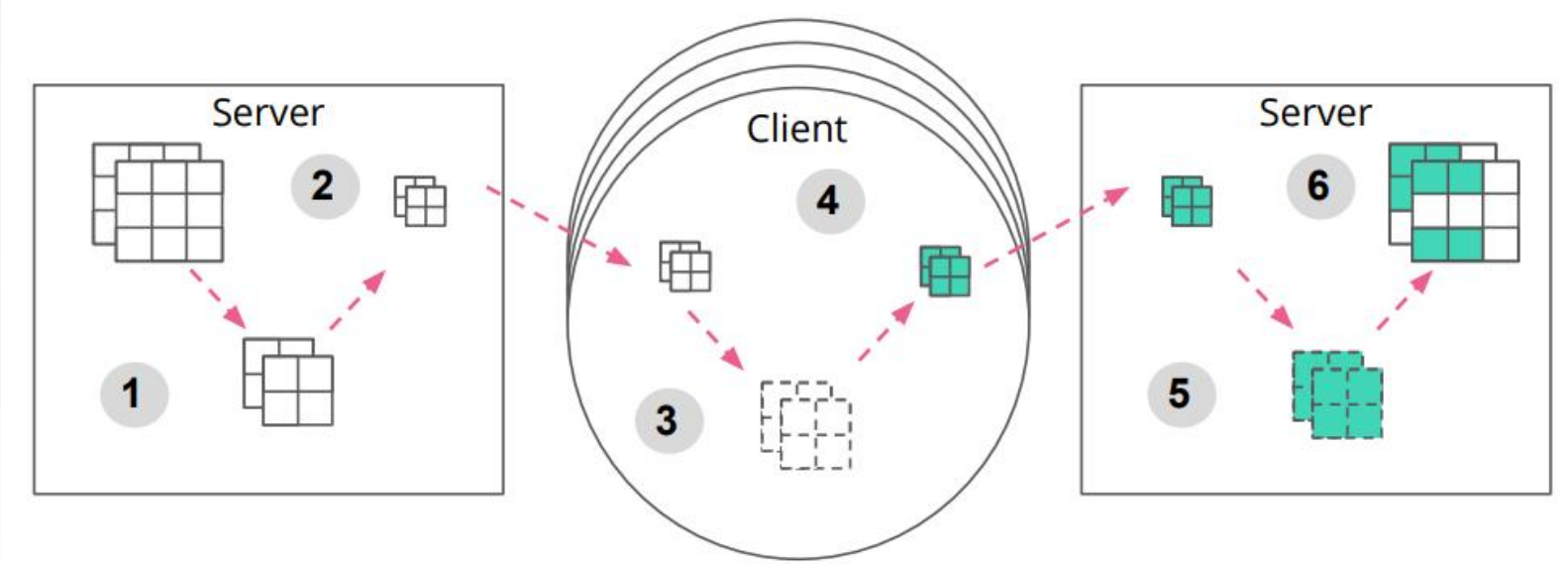
- Mini-batch optimization methods have been shown to have limited flexibility to adapt to communication-computation trade-offs that would maximally leverage distributed data processing.
- Allow for a variable number of local updates to be applied on each machine in parallel at each communication round
- For convex objectives, distributed local-updating primal-dual methods have emerged as a popular way to tackle such a problem.



Communication-efficiency

Compression schemes

- Use lossy compression and dropout to reduce server-to-device communication.



(1) constructing a sub-model via Federated Dropout, and by (2) lossily compressing the resulting object. This compressed model is then sent to the client, who (3) decompresses and trains it using local data, and (4) compresses the final update. This update is sent back to the server, where it is (5) decompressed and finally, (6) aggregated into the global model

Communication-efficiency

Decentralized Training

- In federated learning, a star network (where a central server is connected to a network of devices) is the predominant communication topology.
- Decentralized algorithms can in theory reduce the high communication cost on the central server.

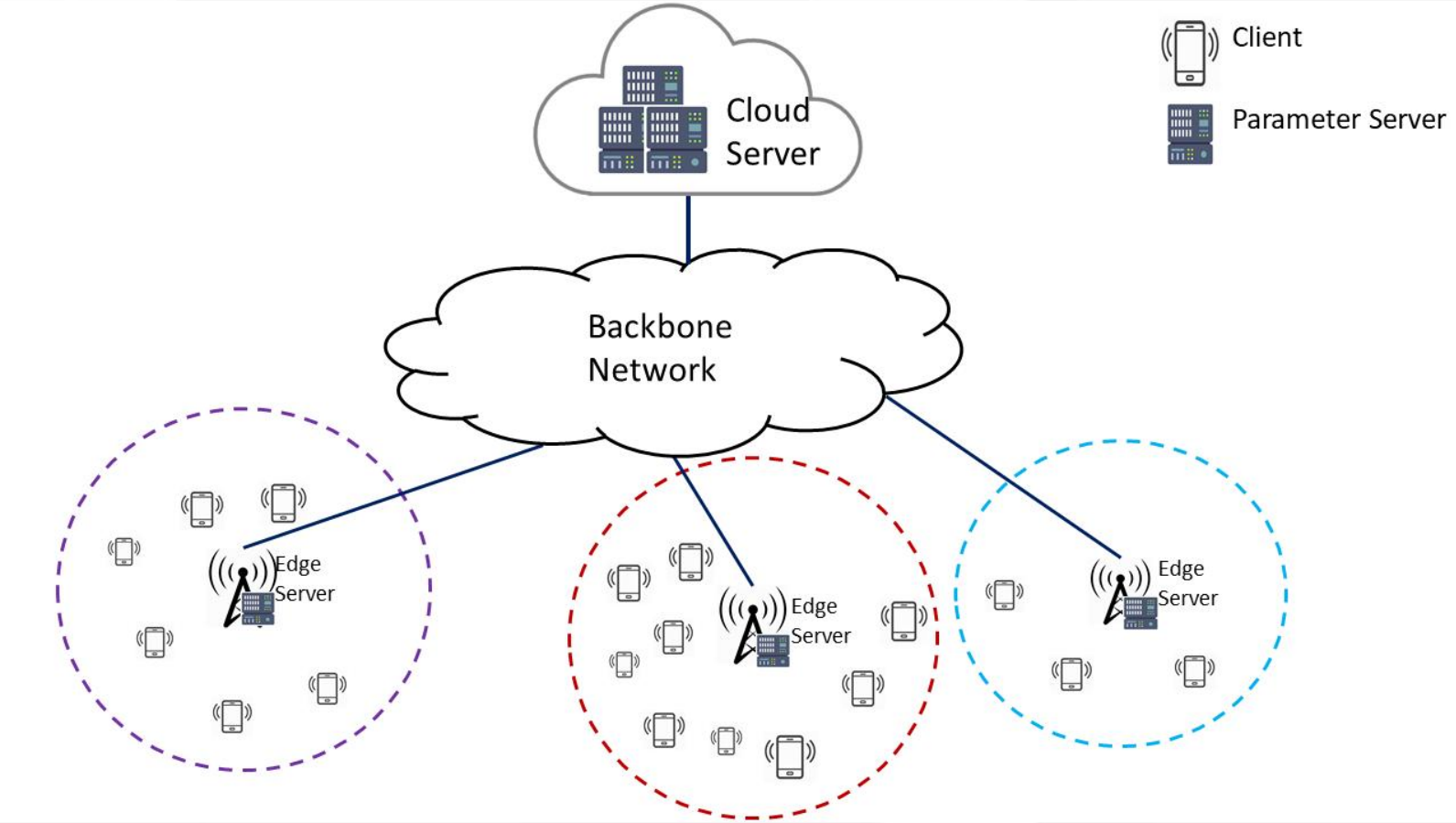
Centralized topology

Decentralized topology

Communication-efficiency

Decentralized Training

- Hierarchical communication patterns.



03

Privacy protection

- Privacy threats/attacks in federated learning (FL)
- Enhance the general privacy-preserving feature of FL
- Associated cost of the privacy-preserving techniques

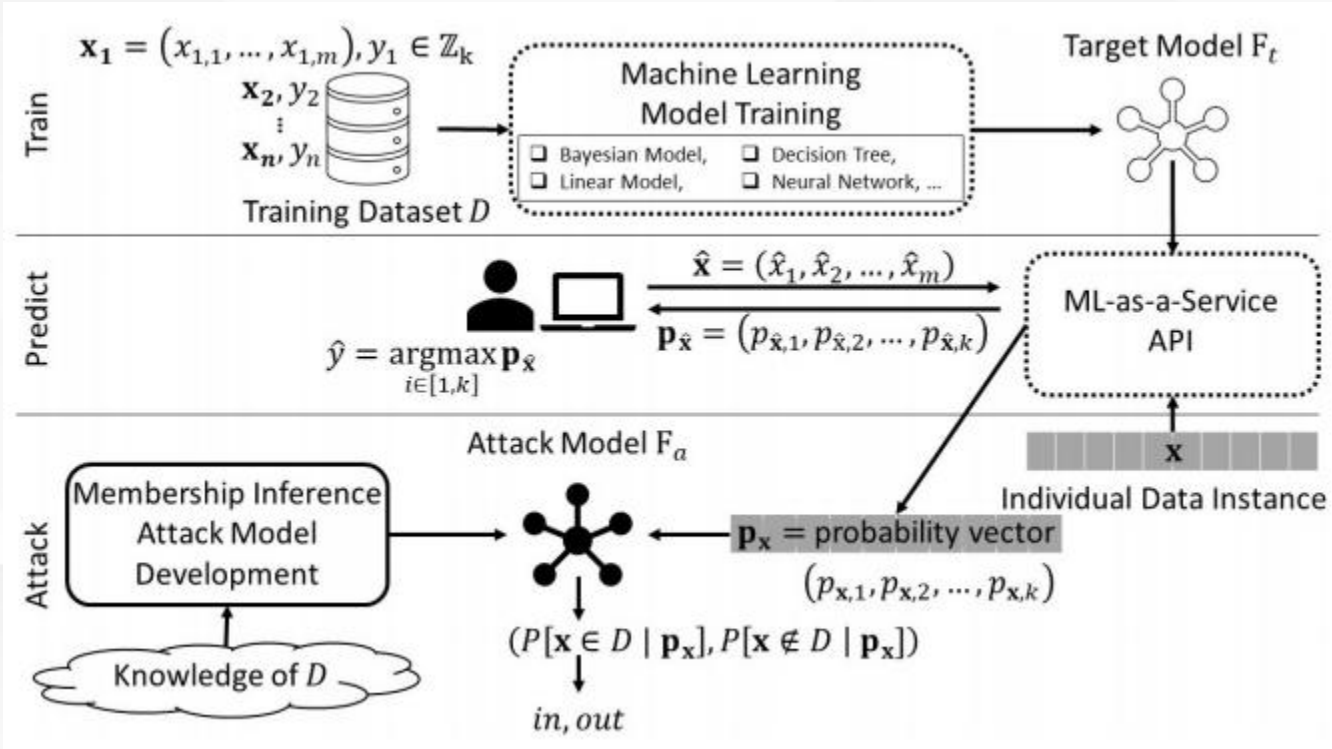
Privacy protection

- Privacy threats/attacks in FL
  - Membership inference attacks
  - Unintentional data leakage and reconstruction through inference
  - GANs-based inference attacks

Privacy protection

Membership inference attacks

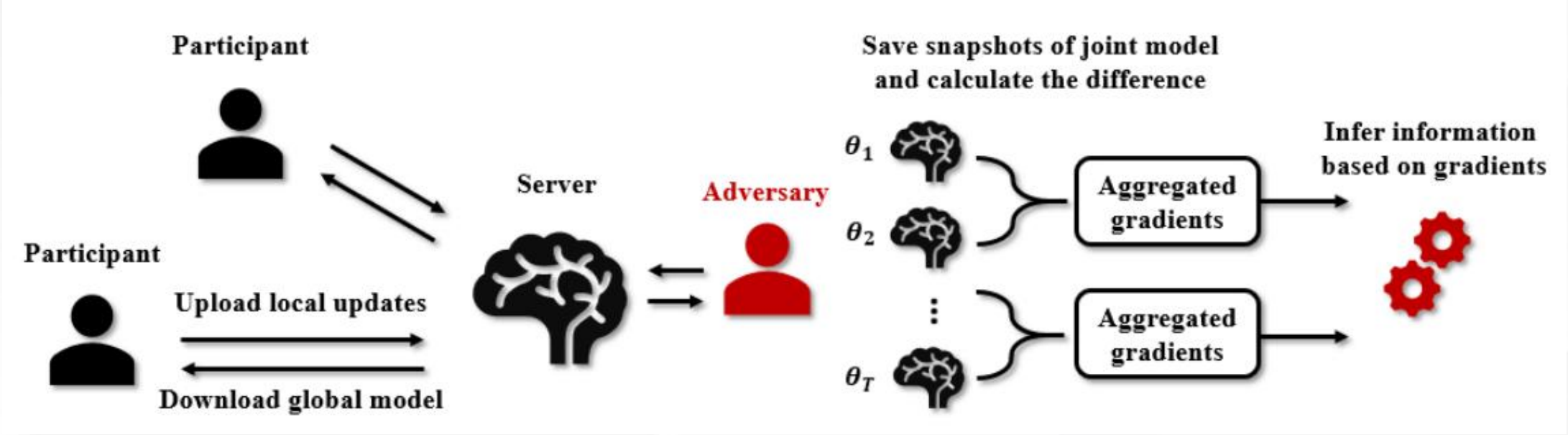
- The neural network is vulnerable to memorize their training data which is prone to passive and active inference attacks.
- The attacker misuses the global model to get information on the training data of the other users.



Privacy protection

Unintentional data leakage and reconstruction

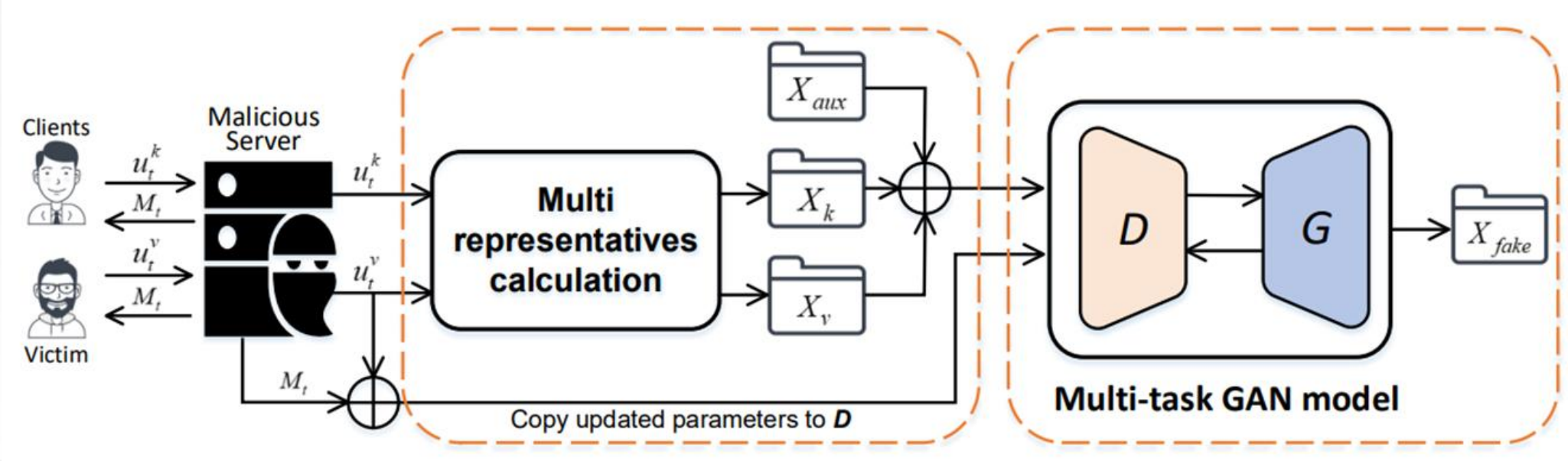
- Is a scenario where updates or gradients from clients leak unintended information at the central server.



Privacy protection

GANs-based inference attacks

- GANs are generative adversarial networks that have gained much popularity in big data domains.
- It is possible to have potential adversaries among FL clients.



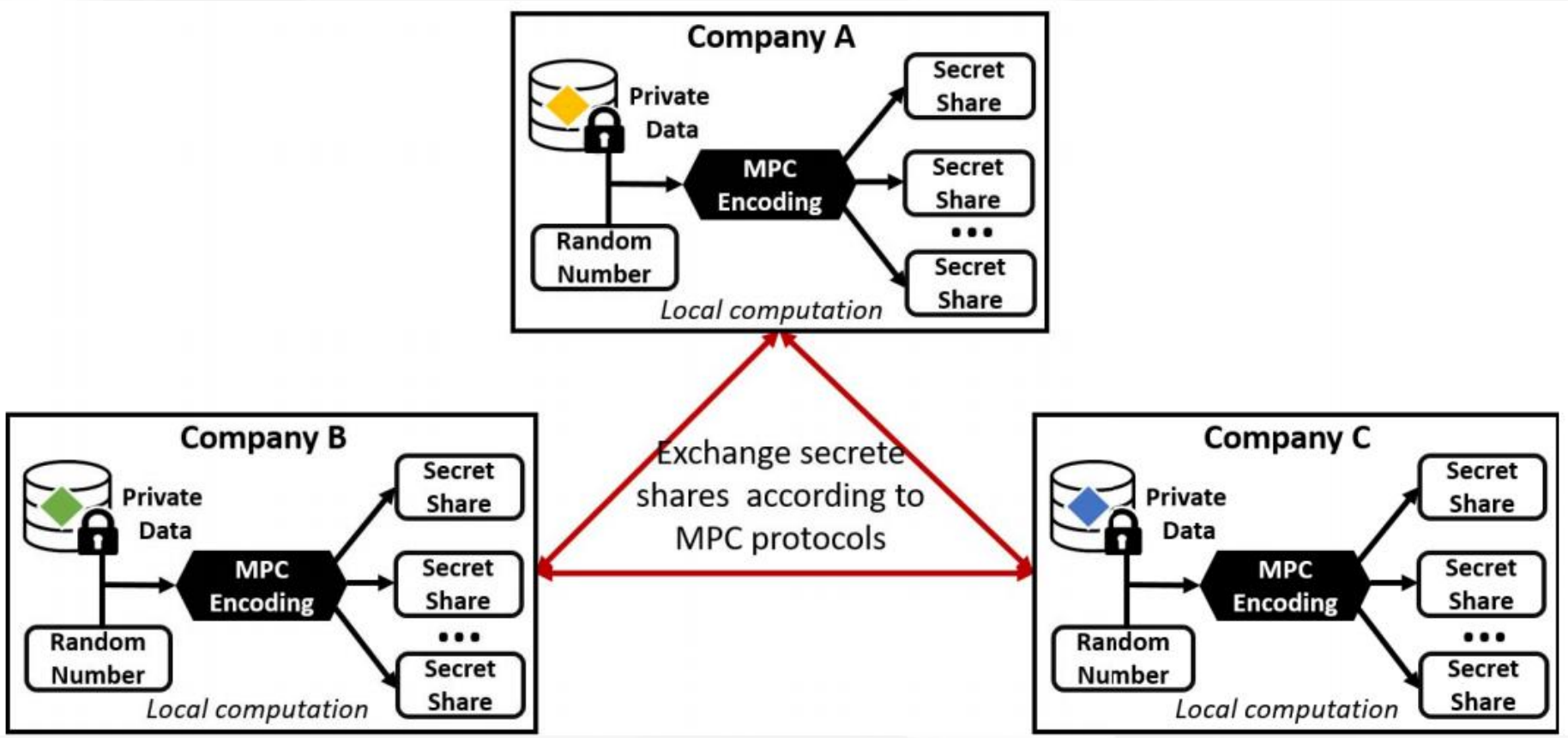
Privacy protection

- Enhance privacy-preserving in FL
  - Secure multi-party computation
  - Differential privacy
  - VerifyNet
  - Adversarial training

Privacy protection

Secure multi-party computation

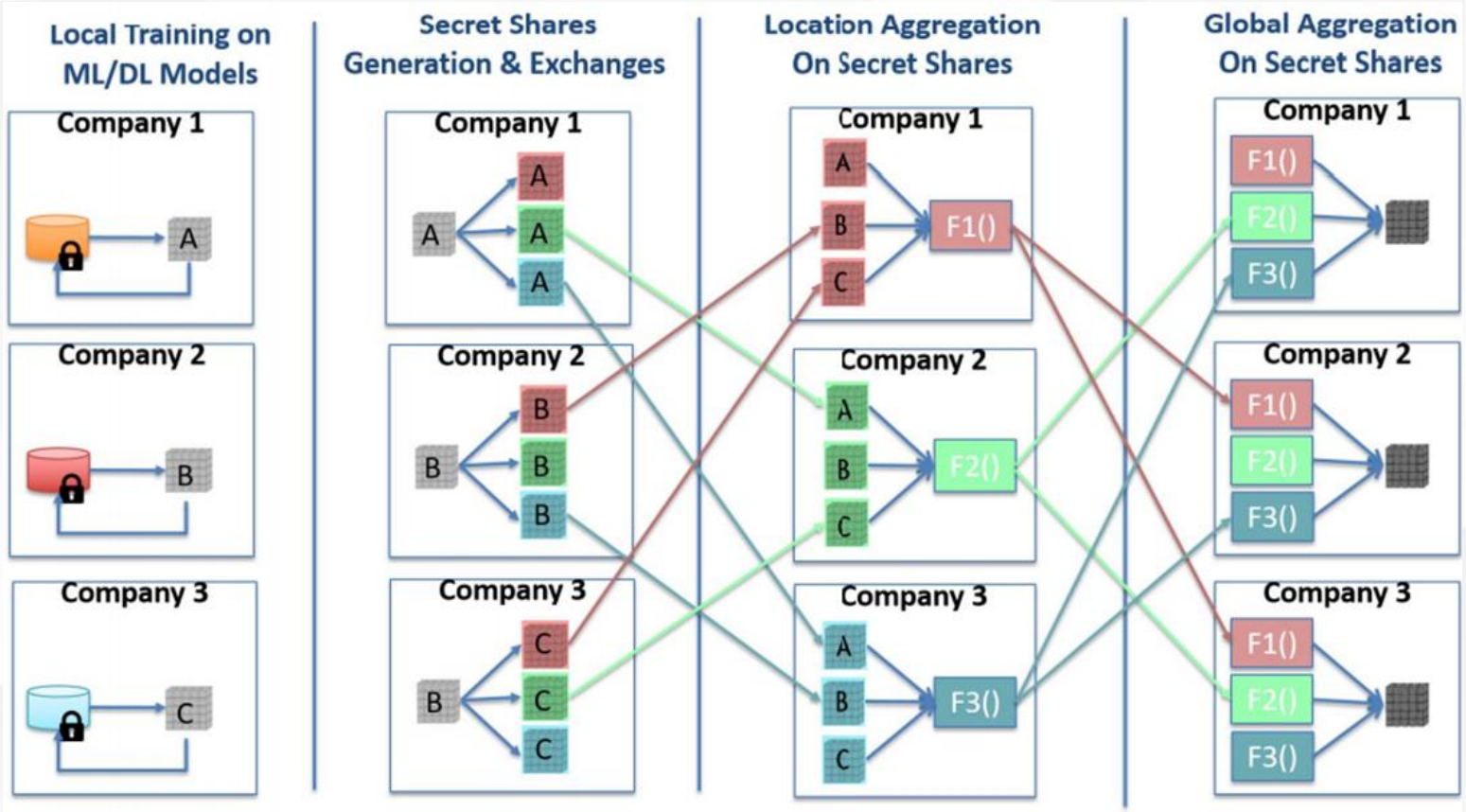
- Secure the inputs of multi-participant while they jointly compute a model or a function.



Privacy protection

Secure multi-party computation

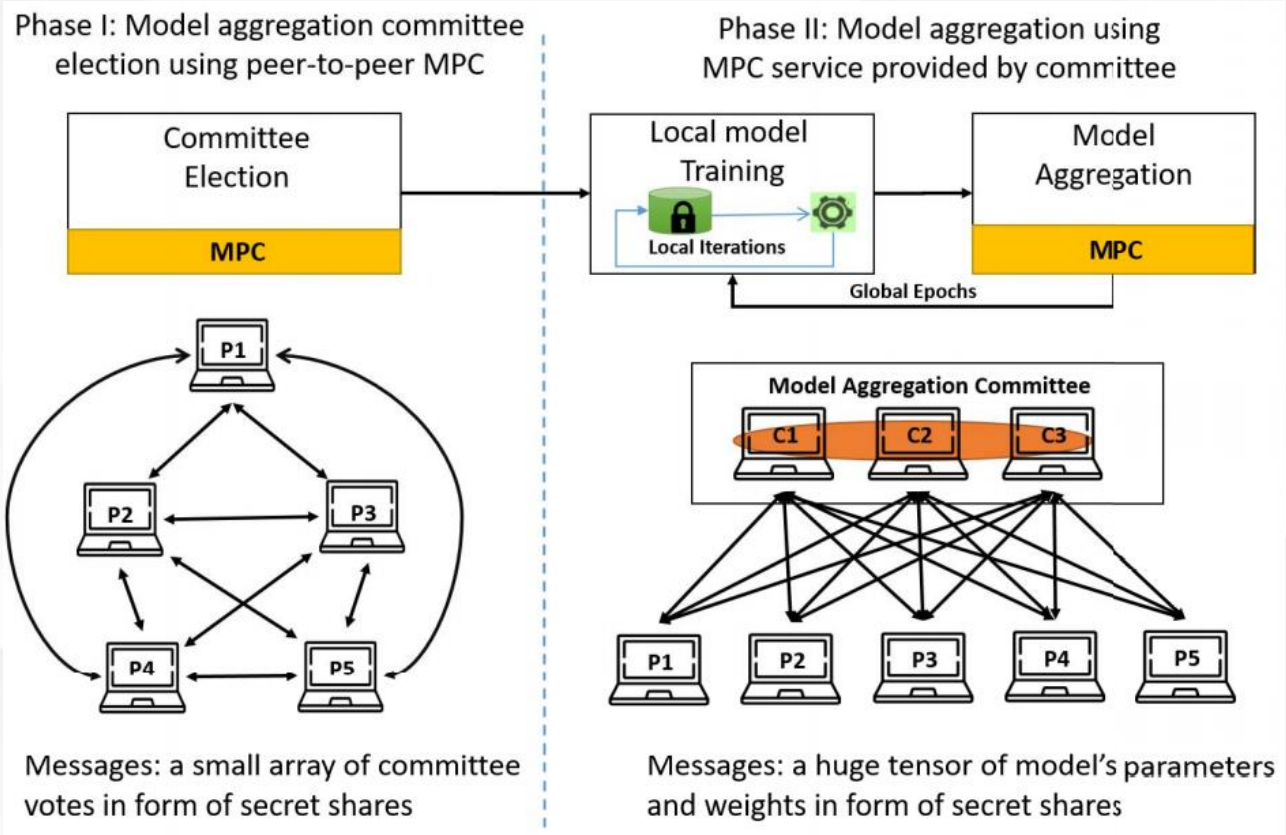
- Secure the inputs of multi-participant while they jointly compute a model or a function.



Privacy protection

Secure multi-party computation

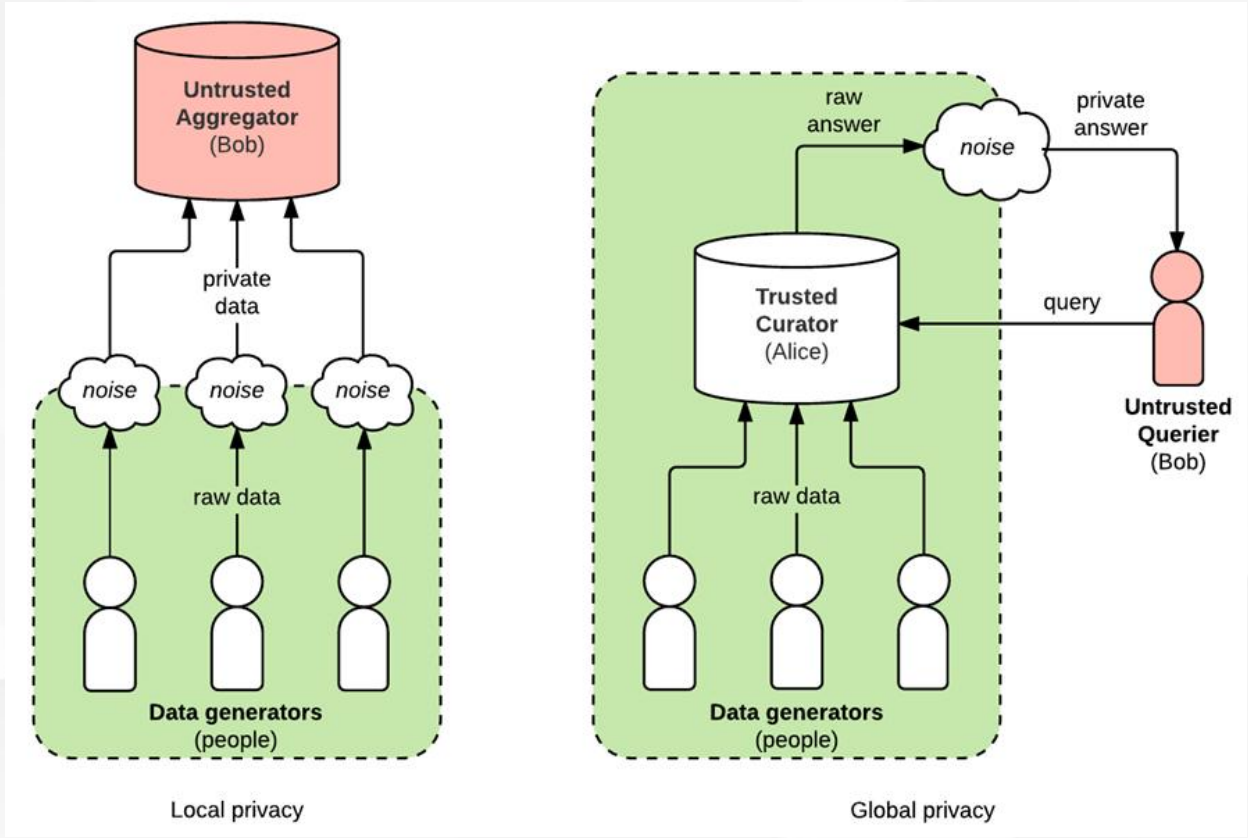
- In FL, the computing efficiency is increased immensely since it only needs to encrypt the parameters instead of the large volume of data inputs.



Privacy protection

Differential Privacy

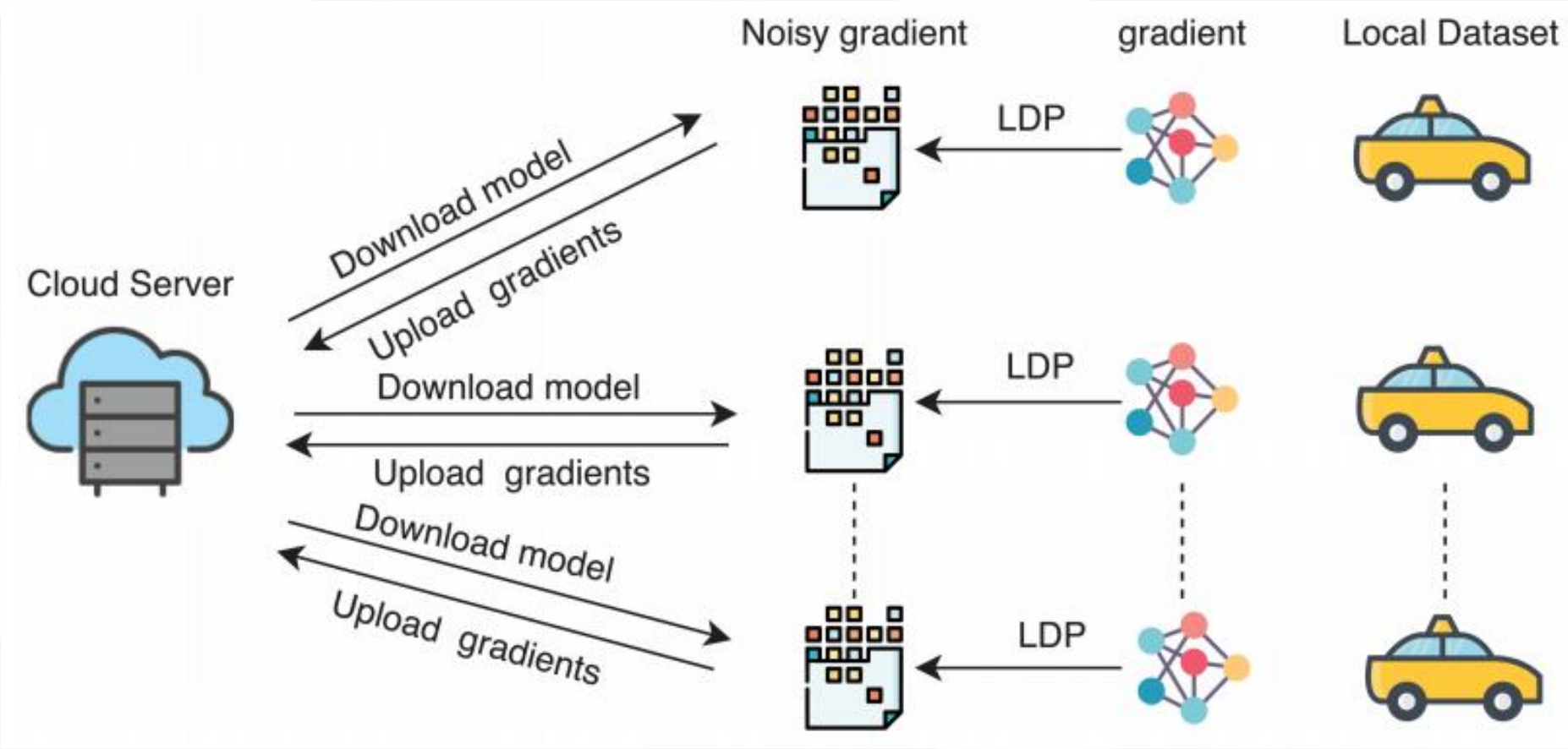
- Add noise to personal sensitive attributes



Privacy protection

Differential Privacy

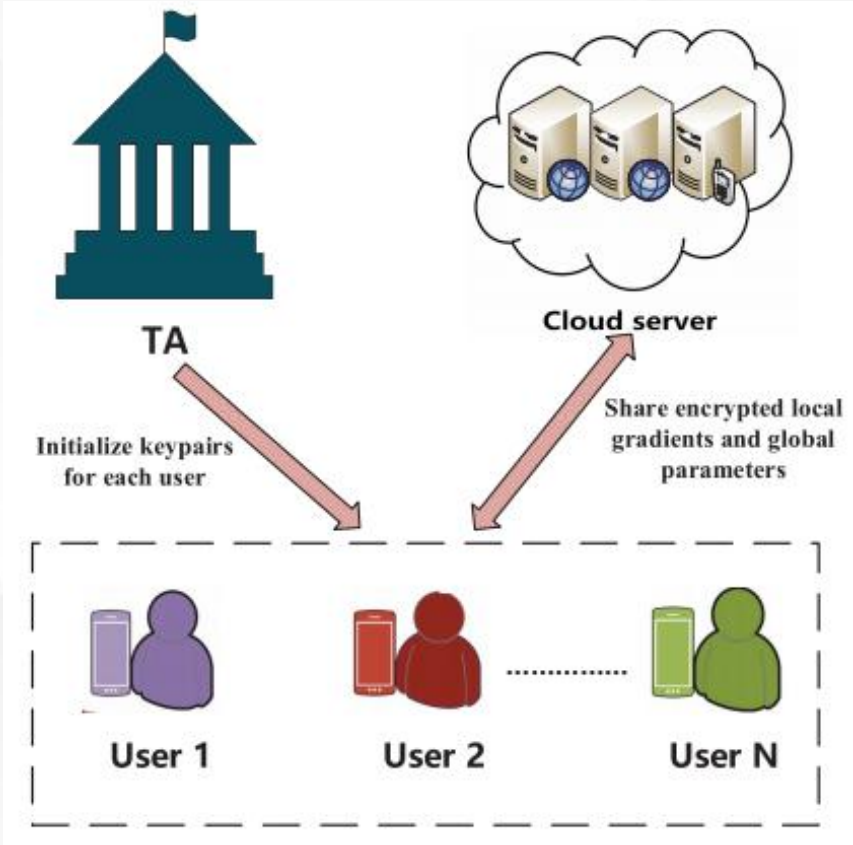
- DP is introduced to add noise to participants' uploaded parameters



Privacy protection

VerifyNet

- It gets listed as a preferred mitigation strategy to preserve privacy as it provides double-masking protocol which makes it difficult for attackers to infer training data.



Privacy protection

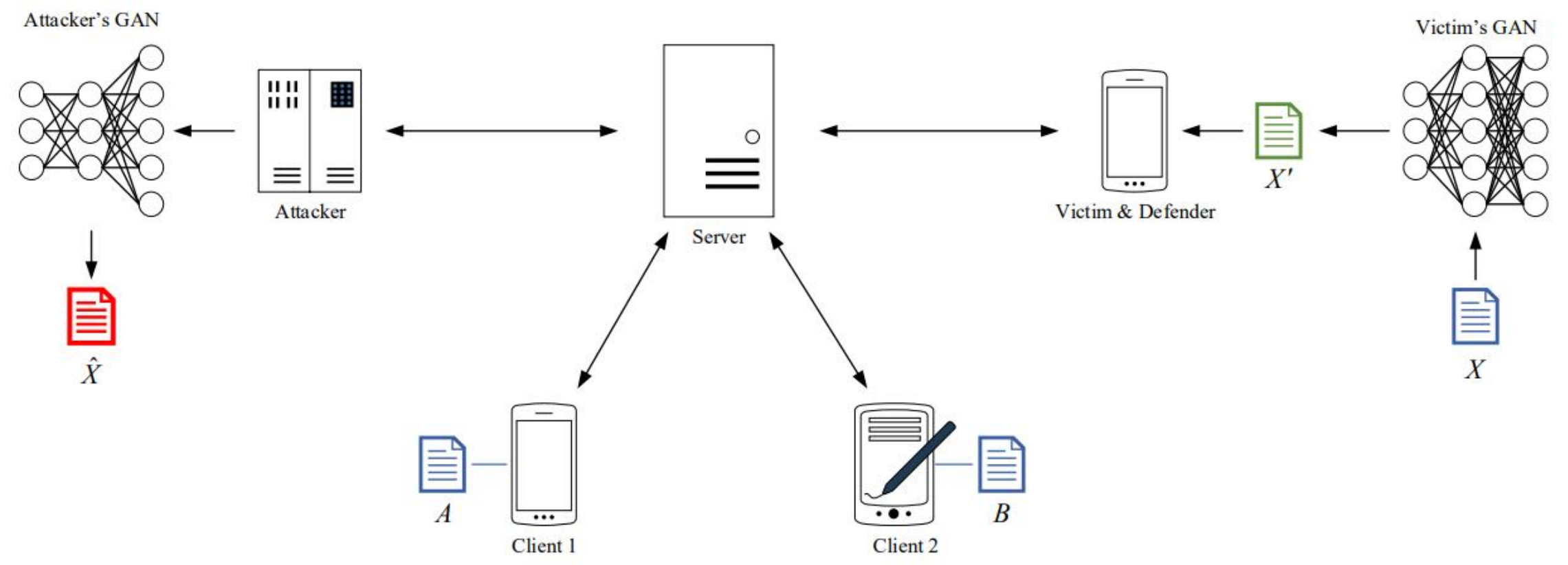
Adversarial training

- Evasion attacks from an adversarial user aims to fool ML models by injecting adversarial samples into the machine learning models.
- The attacker tries to impact the robustness of the FL model with perturbed data.
- Adversarial training, which is a proactive defense technique, tries all permutations of an attack from the beginning of the training phase to make the FL global model robust to known adversarial attacks.

Privacy protection

Adversarial training

- Use GAN to generate fake training data.



Privacy protection

Associated cost

Approach	Cost	Methodology
Secure Multi-party Computation	Efficiency loss due to encryption	Encrypt uploaded parameters
Differential Privacy	Accuracy loss due to added noise in client's model	Add random noise to uploaded parameters
Hybrid	Subdued cost on both efficiency and accuracy	Encrypt the manipulated parameter
VerifyNet	Communication overhead	Double-masking protocol Verifiable aggregation results
Adversarial Training	Computation power, training time for adversarial samples	Include adversarial samples in training data

04

Heterogeneity

- Addressing systems heterogeneity
- Addressing statistical heterogeneity

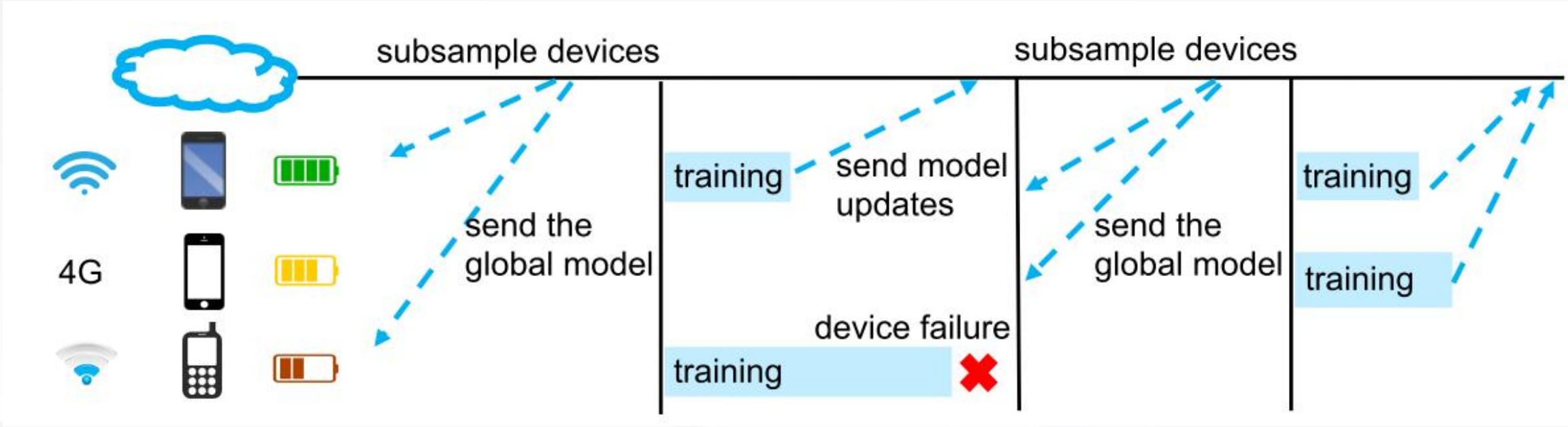
Heterogeneity

- Addressing systems heterogeneity
  - Asynchronous communication
  - Active sampling
  - Fault tolerance
  - Using client-specific model

Heterogeneity

Asynchronous communication

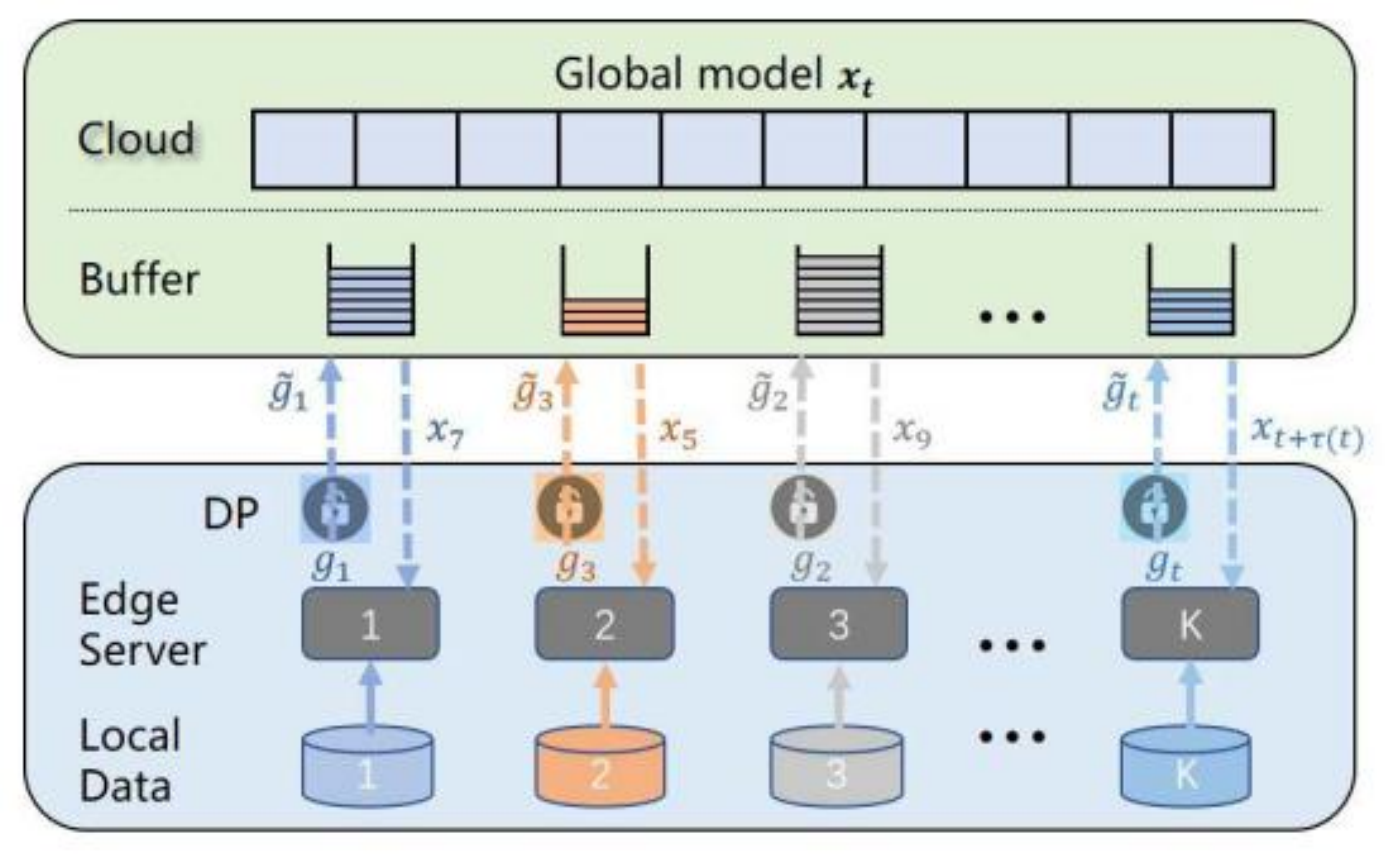
- Synchronous schemes are simple and guarantee a serial-equivalent computational model, but they are also more susceptible to stragglers in the face of device variability.



Heterogeneity

Asynchronous communication

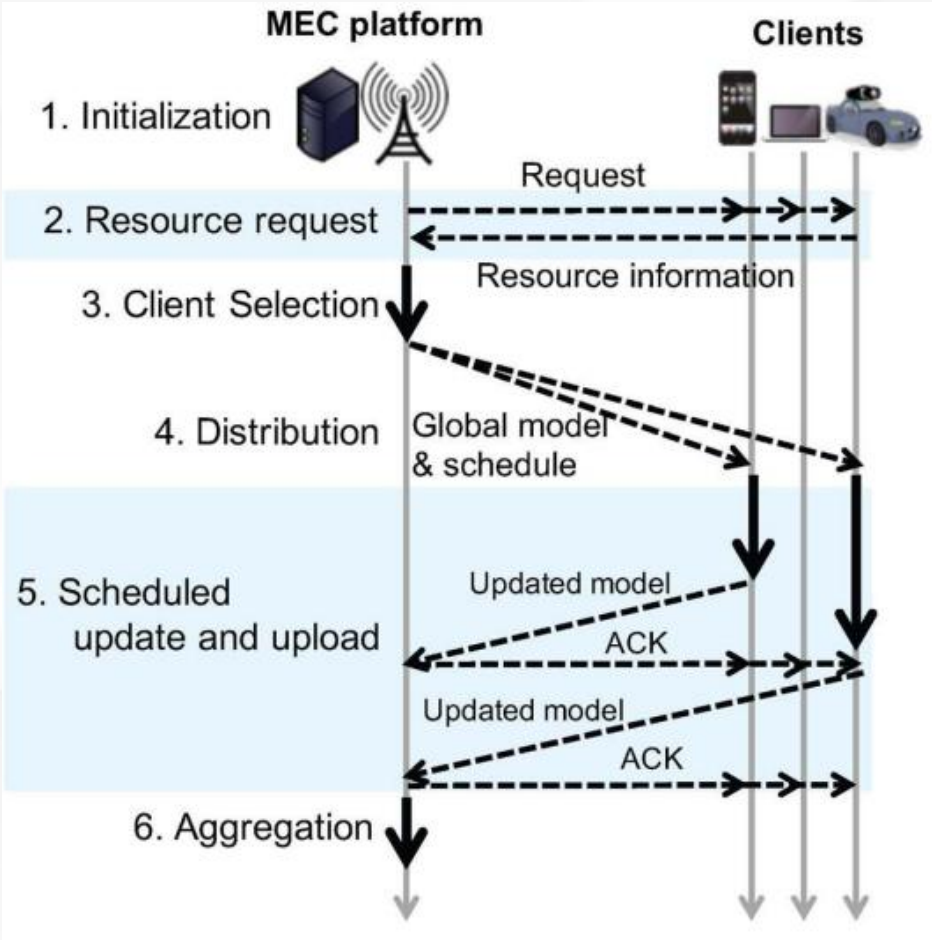
- Asynchronous schemes are an attractive approach to mitigate stragglers in heterogeneous environments.



Heterogeneity

Active sampling

- Actively selecting participating devices at each round.



Heterogeneity

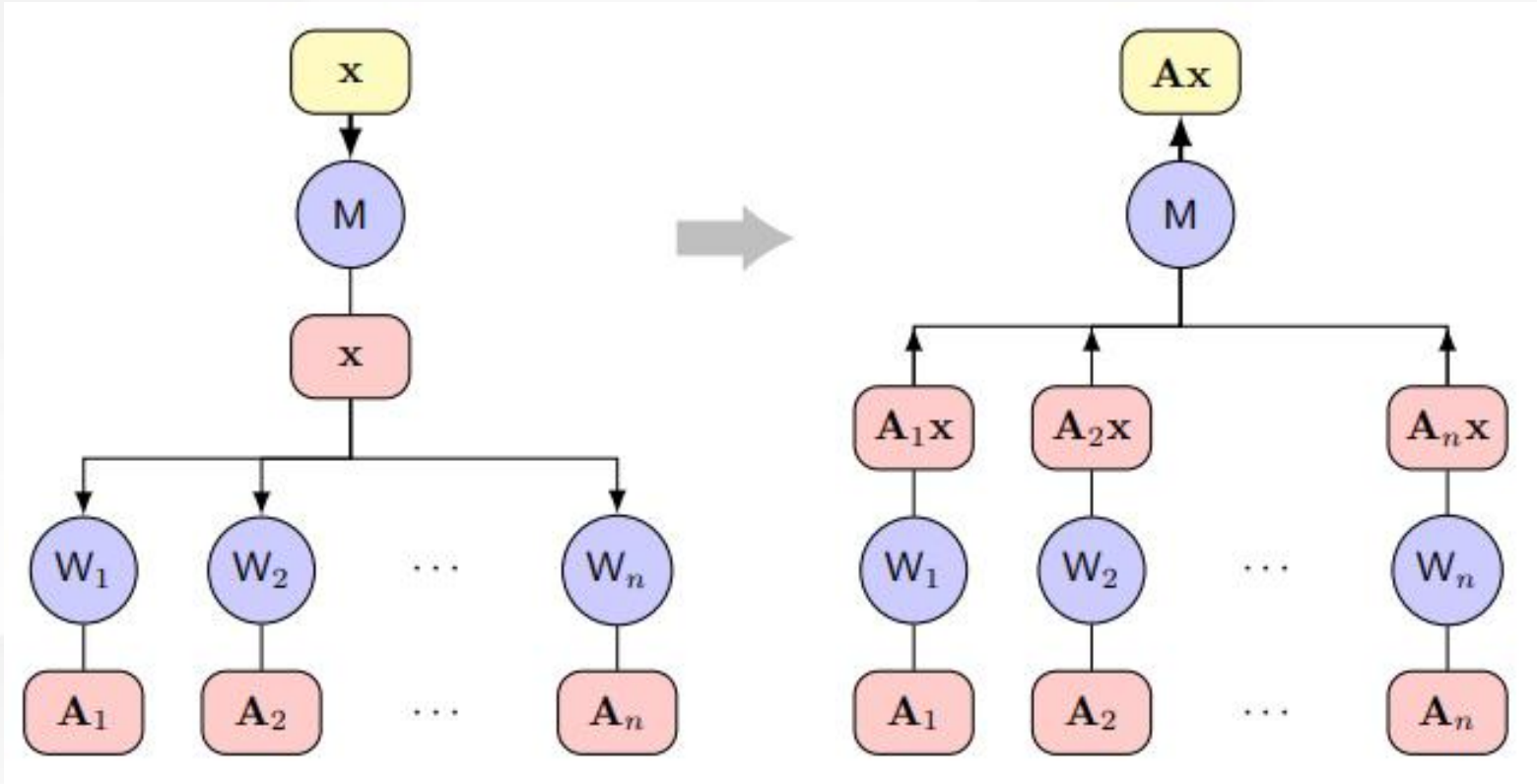
Fault tolerance

- Fault tolerance has been extensively studied in the systems community and is a fundamental consideration of classical distributed systems.
- When learning over remote devices, however, fault tolerance becomes more critical.
- One practical strategy is to simply ignore such device failure, which may introduce bias into the device sampling scheme if the failed devices have specific data characteristics.

Heterogeneity

Fault tolerance

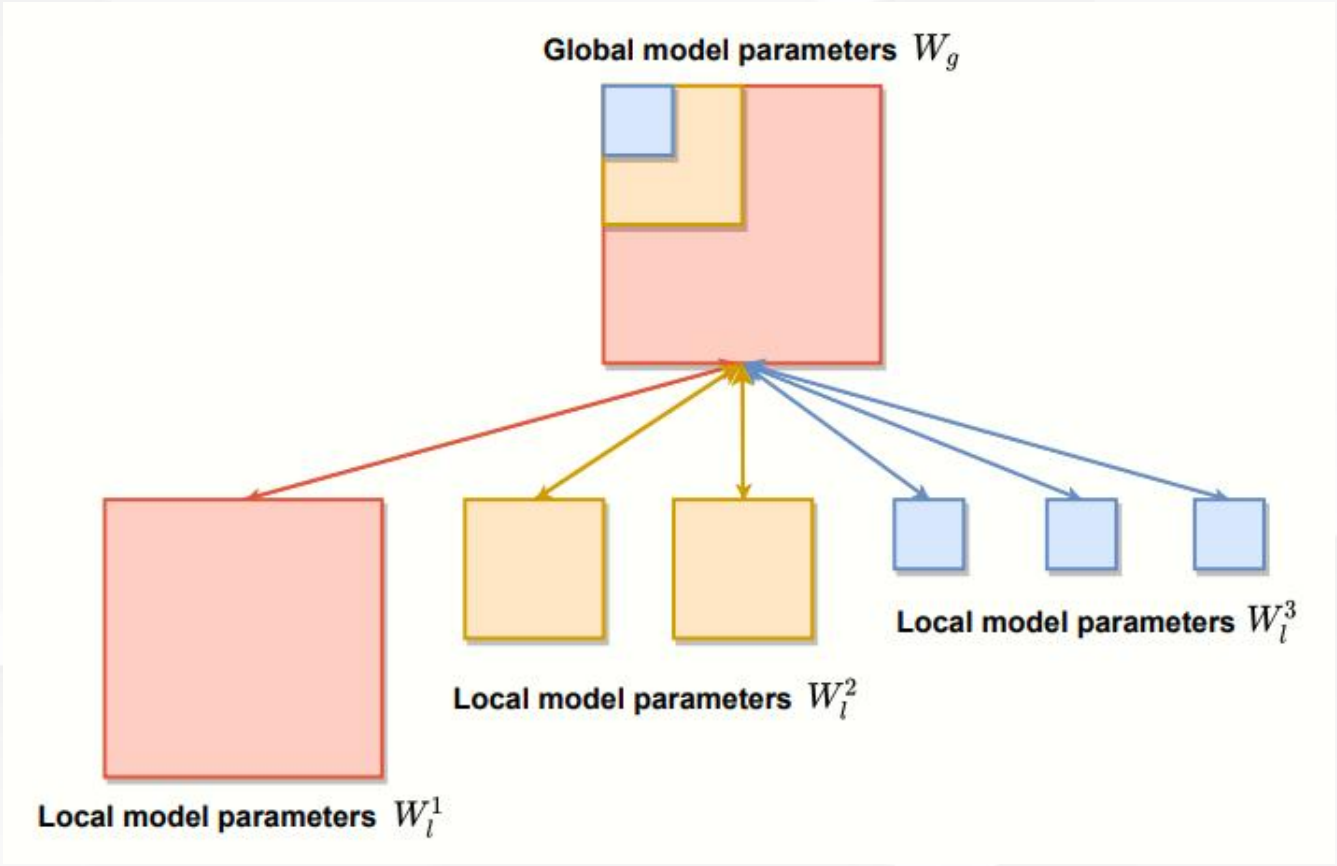
- Coded computation is another option to tolerate device failures by introducing algorithmic redundancy.



Heterogeneity

Using client-specific model

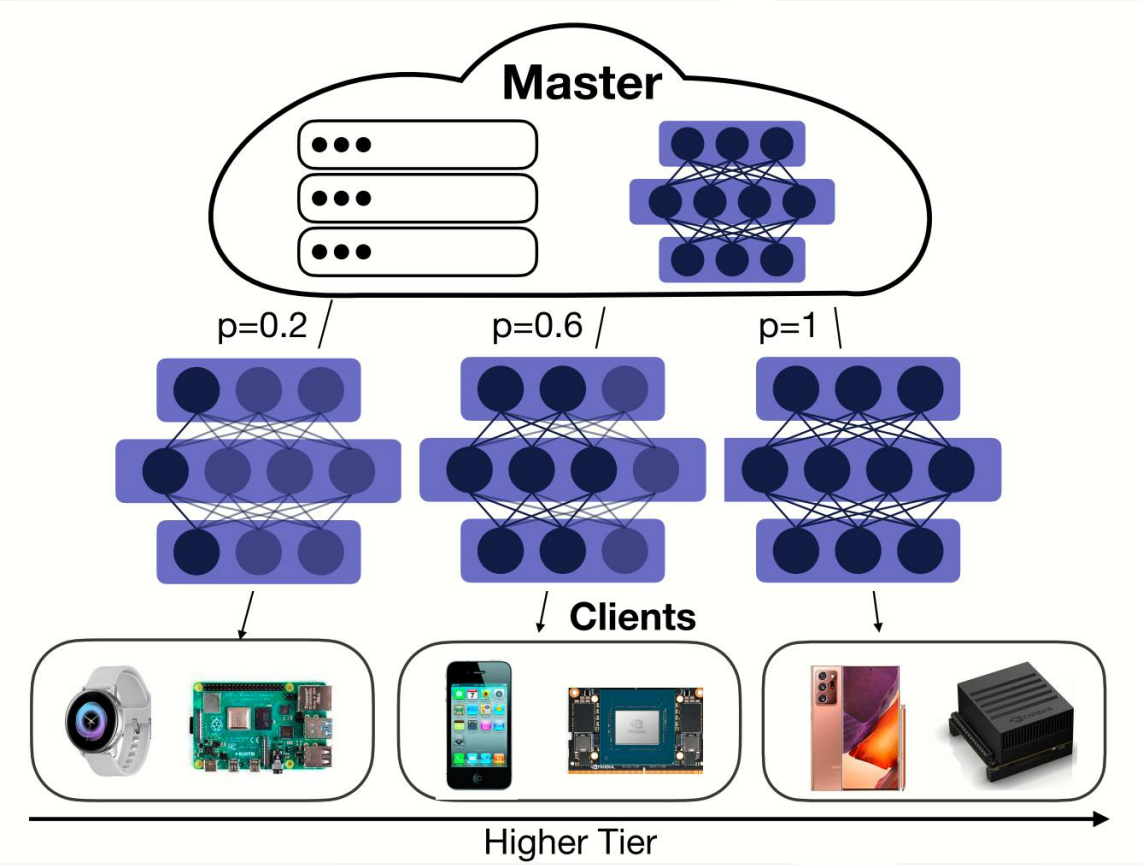
- HeteroFL trains heterogeneous local models and aggregate them stably and effectively into a single global inference model.



Heterogeneity

Using client-specific model

- FjORD employs Ordered Dropout (OD) to tailor the amount of computation to the capabilities of each participating device.

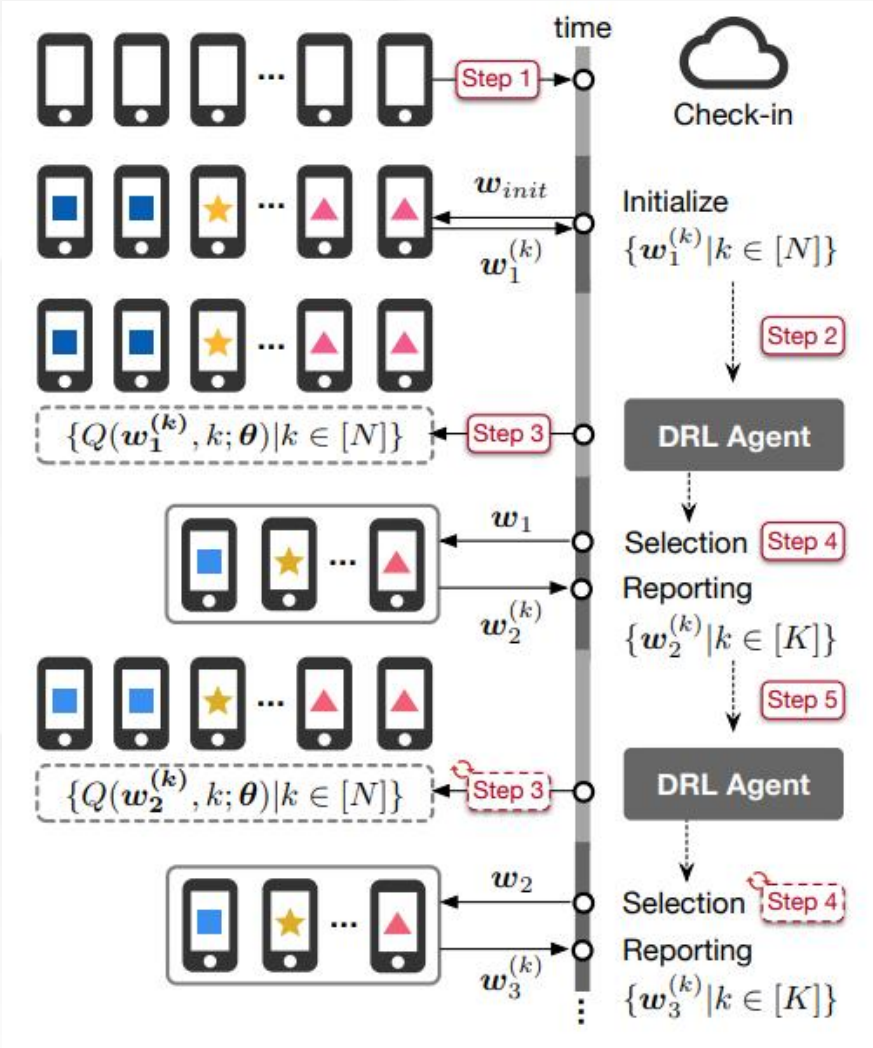


Heterogeneity

- Statistical heterogeneity
  - Overcome the non-IID and unbalanced issue
  - Utilize the non-IID and unbalanced characteristic

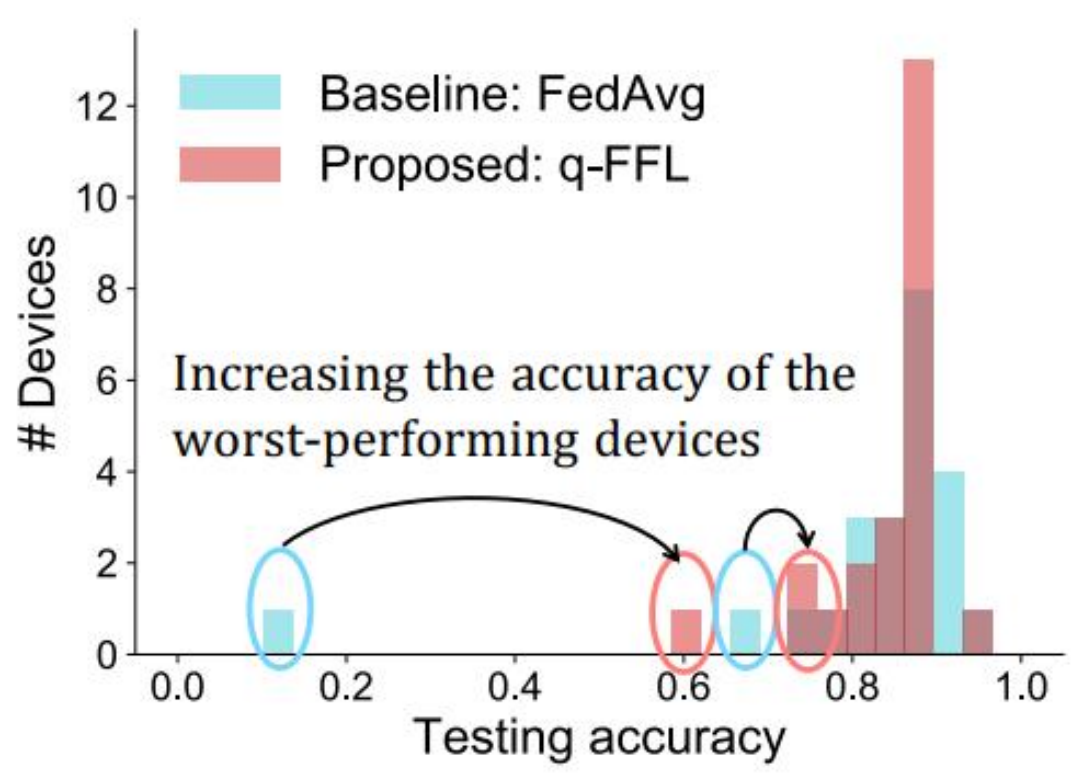
Heterogeneity

- Overcome the non-IID and unbalanced issue
  - Although the data is not independent and identically distributed among all the clients, we can relieve this issue by client selection.
  - Client selection can be formulated as a deep reinforcement learning problem in federated learning.
  - It solely relies on model weight information to determine which device may improve the global model the most —thus preserving the same level of privacy as the original FL does.



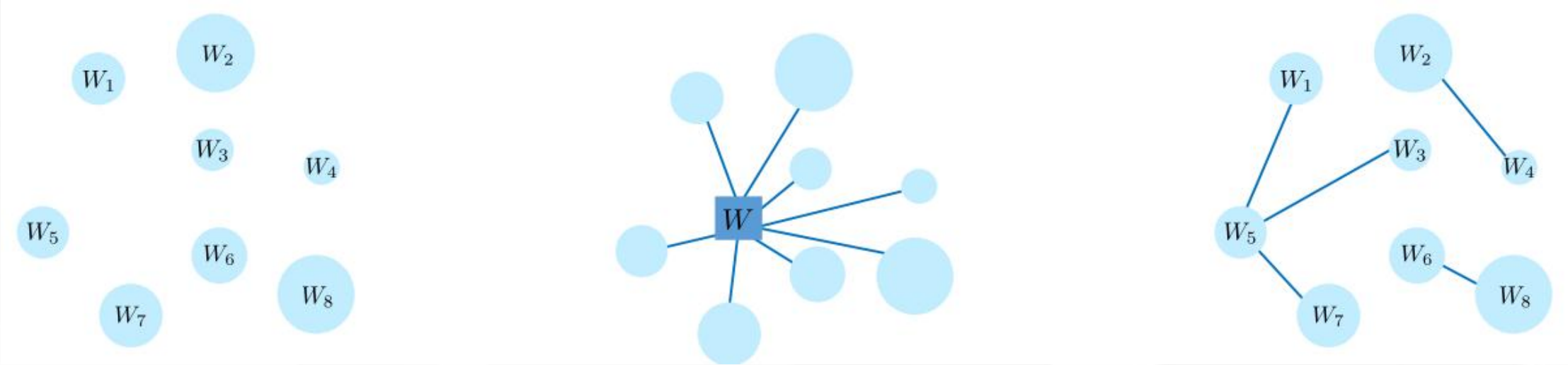
Heterogeneity

- Overcome the non-IID and unbalanced issue
  - Devices with higher loss are given higher relative weight to encourage less variance in the final accuracy distribution.



Heterogeneity

- **Utilize the non-IID and unbalanced characteristic**
  - Non-IID data is not just an issue for federated learning, but also a natural feature in this setting.
  - Personalized federated learning is welcomed.



Learn personalized models for each device; do not learn from peers.

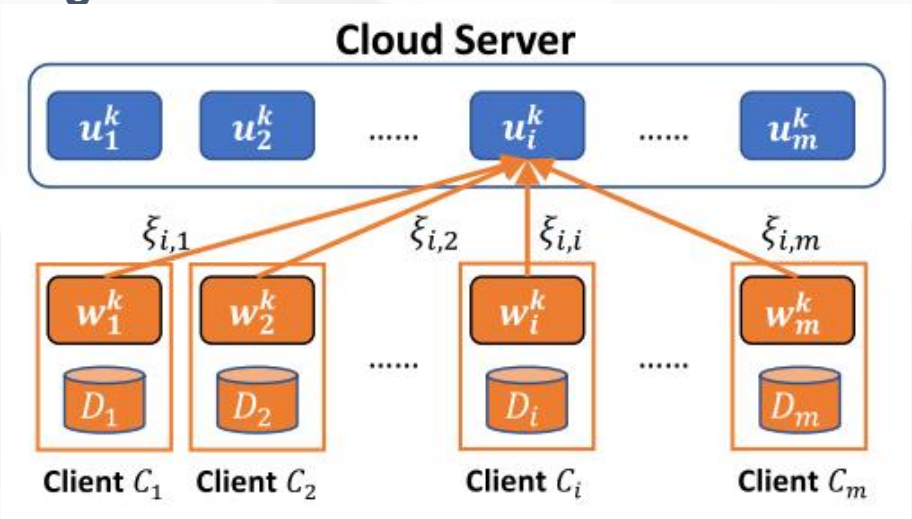
Learn a global model; learn from peers.

Learn personalized models for each device; learn from peers.

Heterogeneity

Personalized federated learning

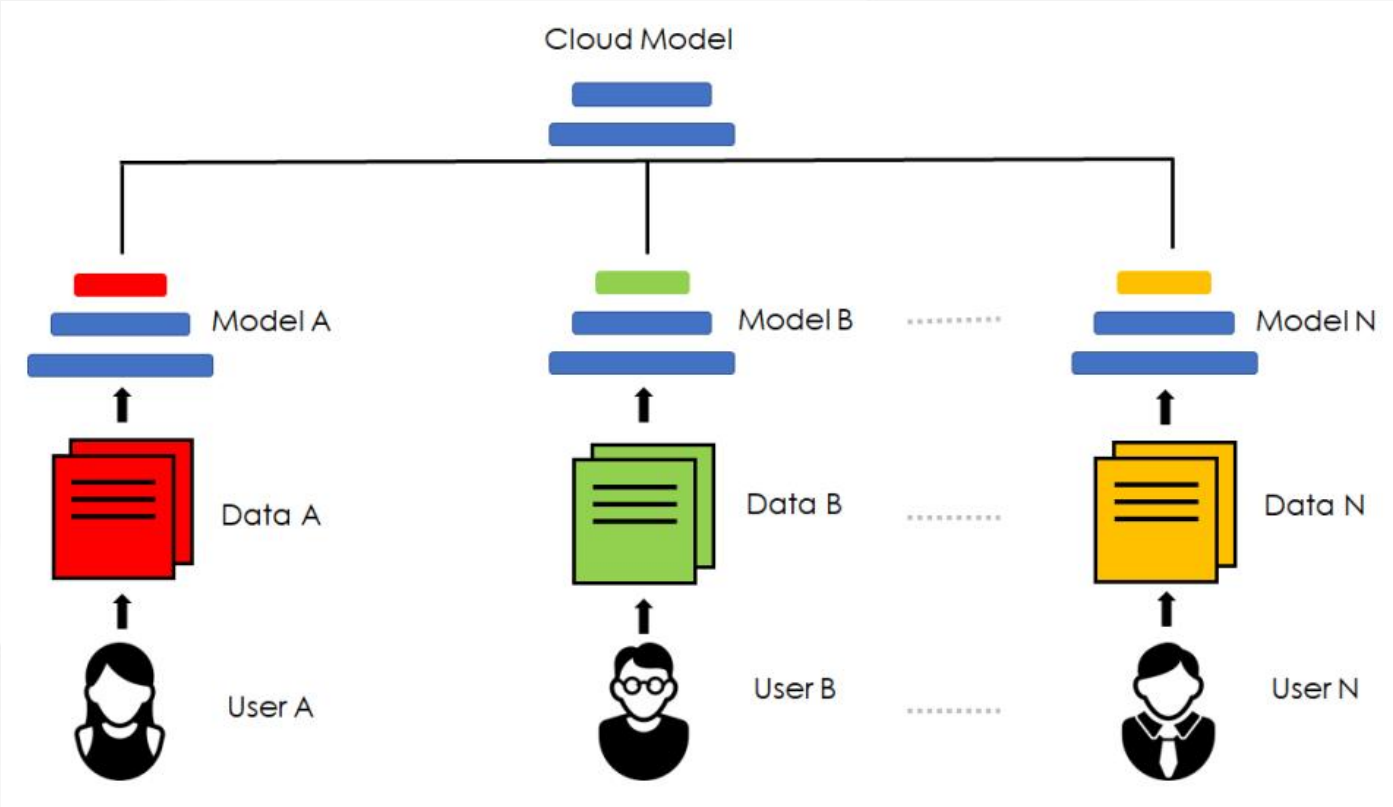
- FedAMP allows each client to own a local personalized model, it maintains a personalized cloud model on the cloud server for each client.
- FedAMP realizes the attentive message passing mechanism by attentively passing the personalized model of each client as a message to the personalized cloud models with similar model parameters.
- FedAMP updates the personalized cloud model of each client by a weighted convex combination of all the messages it receives.



Heterogeneity

Personalized federated learning

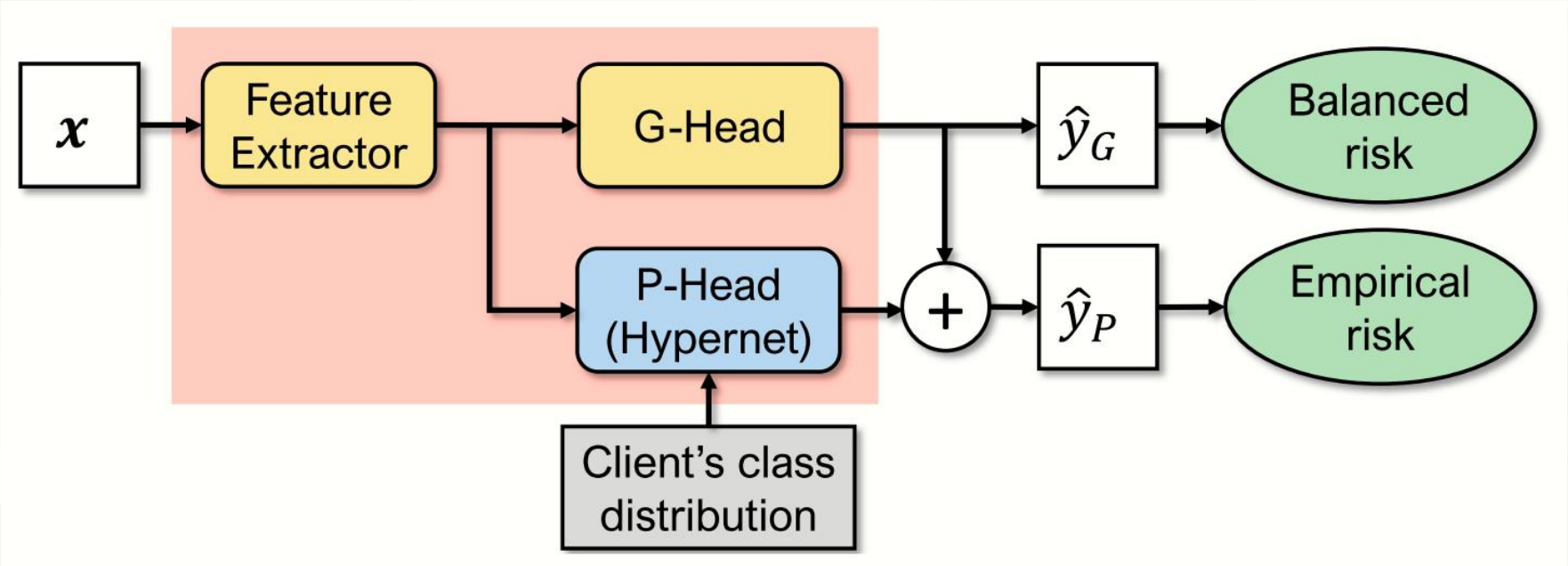
- The base layers are shared with the parameter server while the personalization layers are kept private by each device.



Heterogeneity

Personalized federated learning

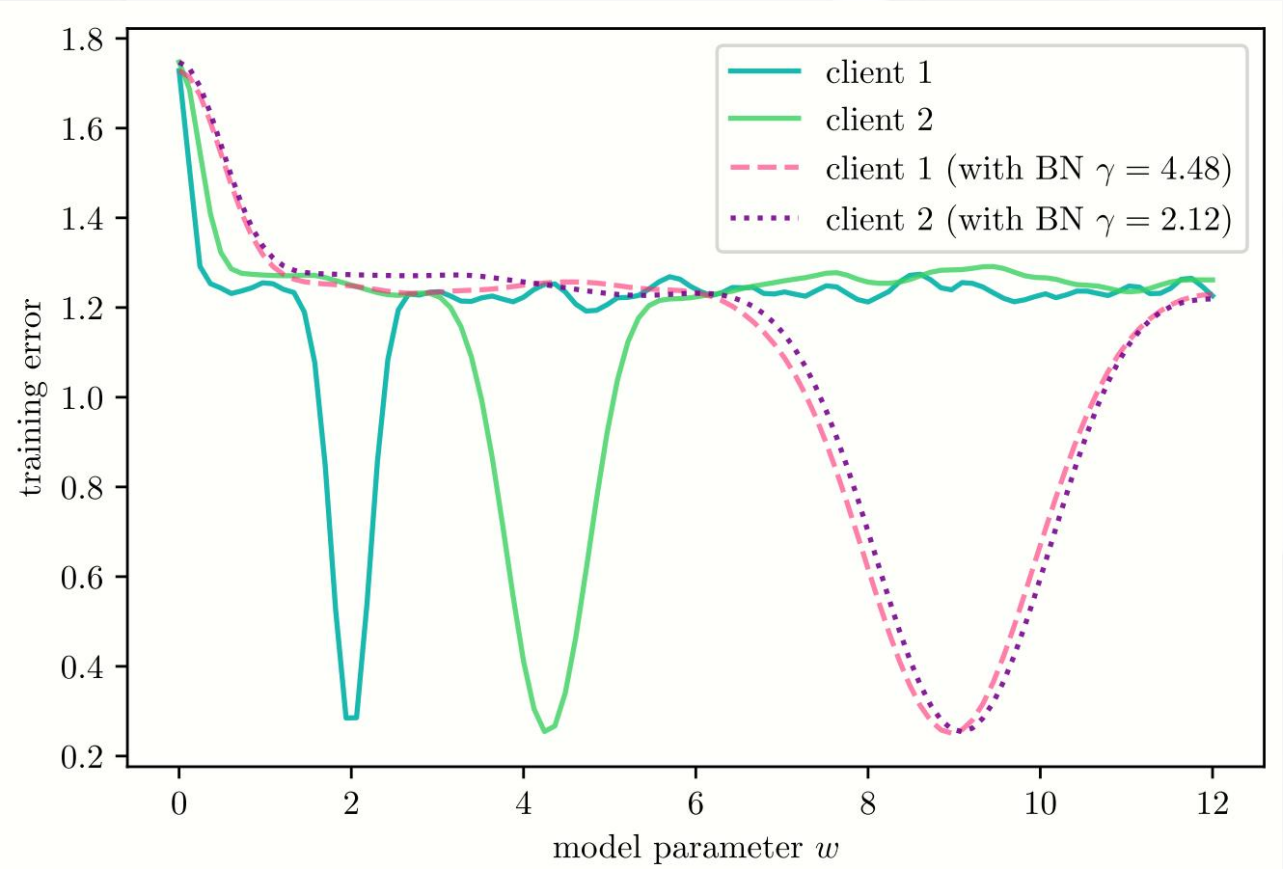
- In addition to only learning for the local objective, FedRoD also proposes to simultaneously learn the balanced objective and the local objective on each client.



Heterogeneity

Personalized federated learning

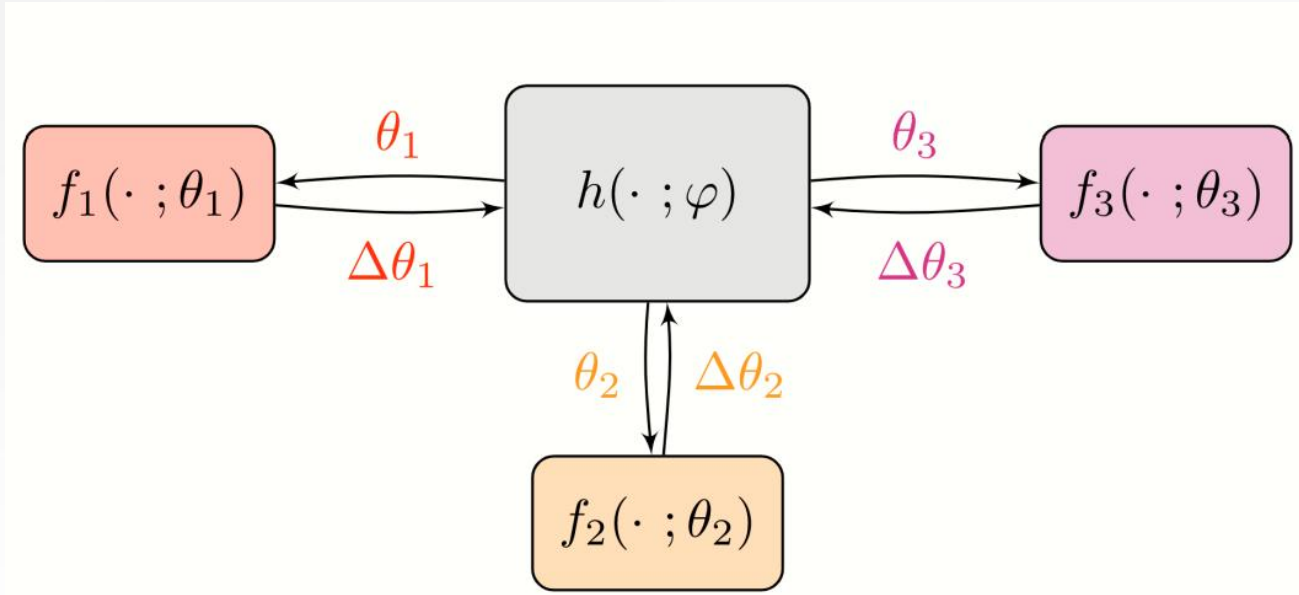
- Rather than localize only the higher layers, FedBN finds that the batch normalization (BN) layers in the ResNets are not beneficial for aggregation and proposes to localize all of them.



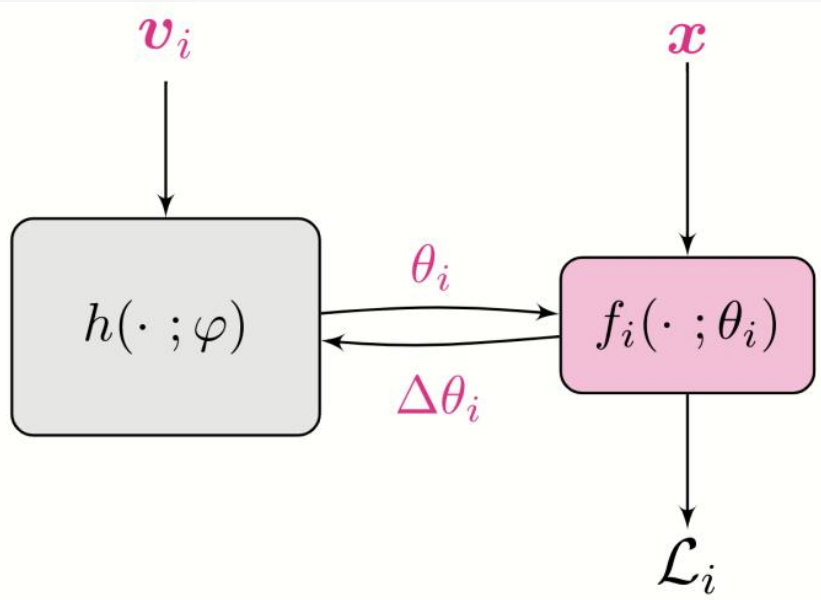
Heterogeneity

Personalized federated learning

- To further personalize models, FedHN assigns one client embedding for one client, and generate client model parameters through the hypernetwork on the server.



Updating the hypernetwork by client models

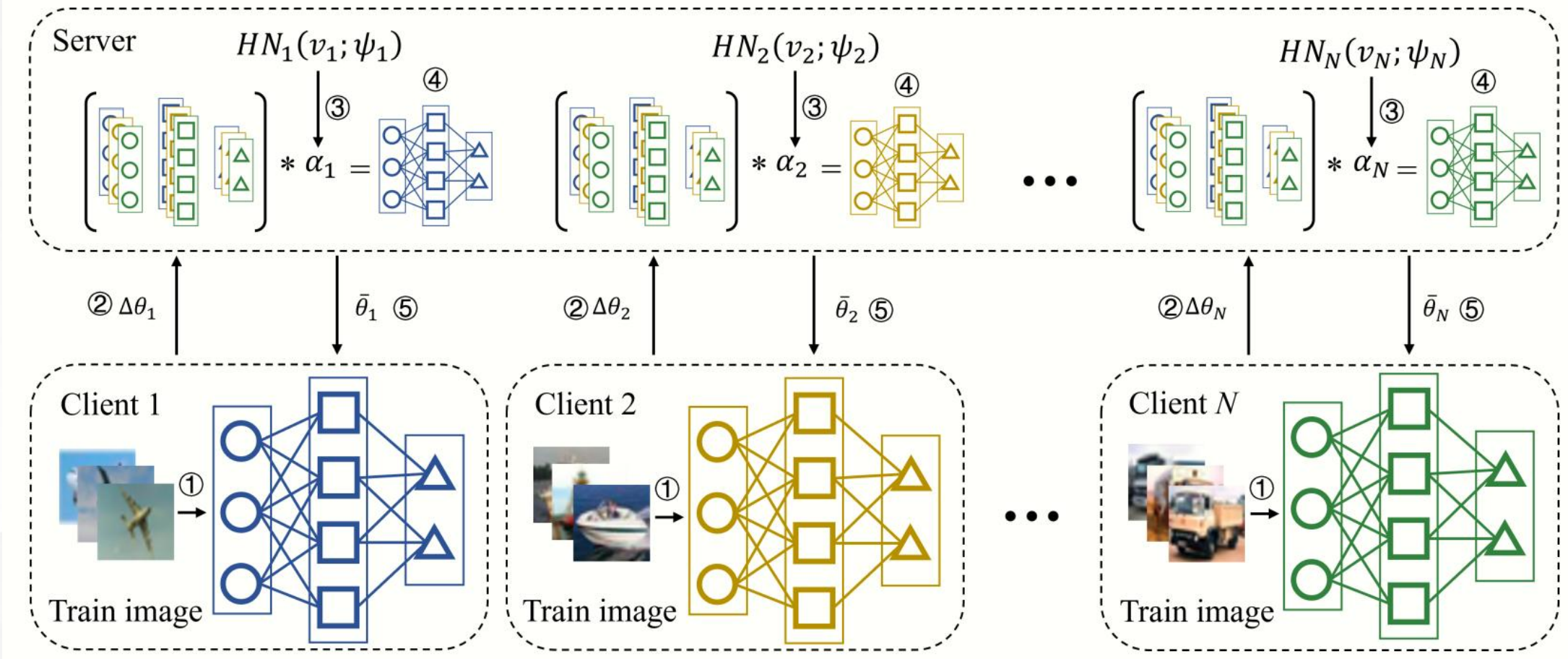


Generating client models

Heterogeneity

Personalized federated learning

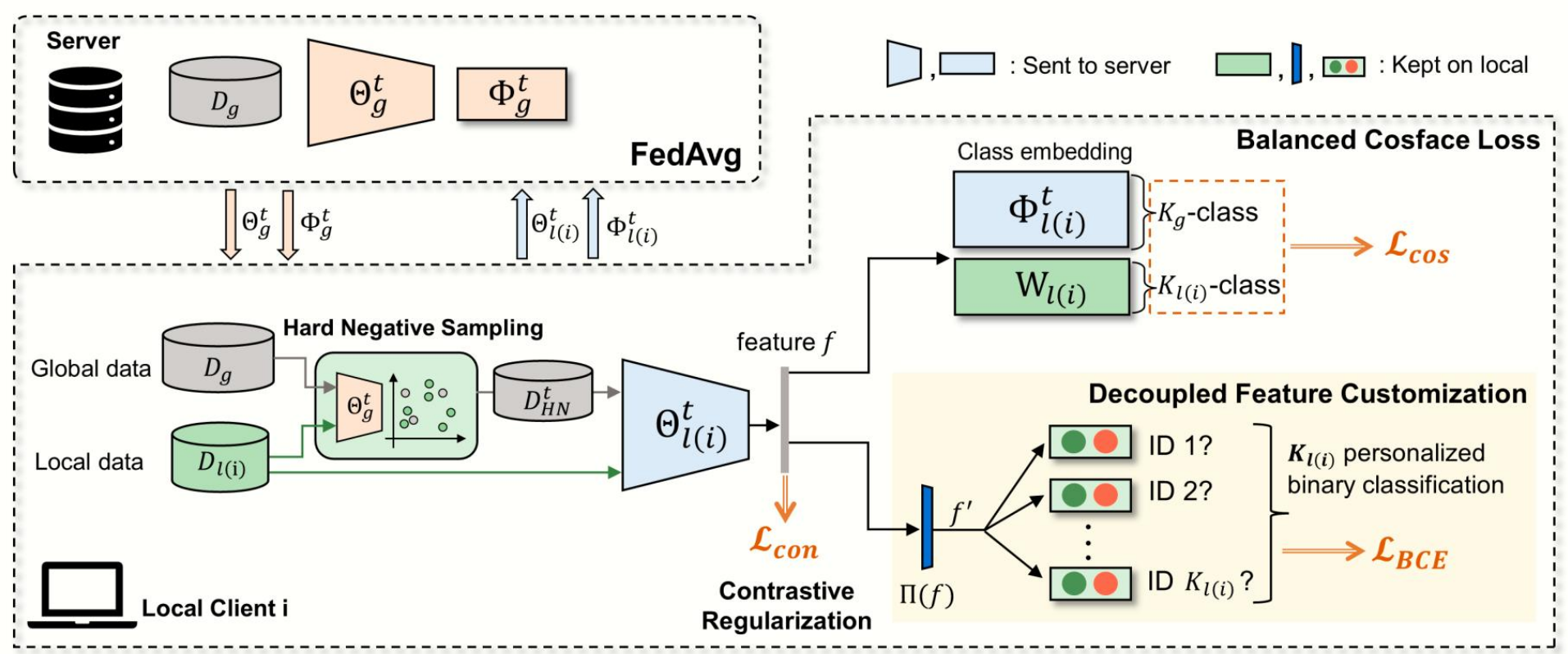
- With the client embeddings, pFedLA generate the layer-wise aggregation weights instead of directly generating the model parameters.



Heterogeneity

Personalized federated learning

- Public dataset also helps FL
- Like FedRoD , FedFR learns two objectives on the client, one of which is the balanced objective, in the face recognition task.



05

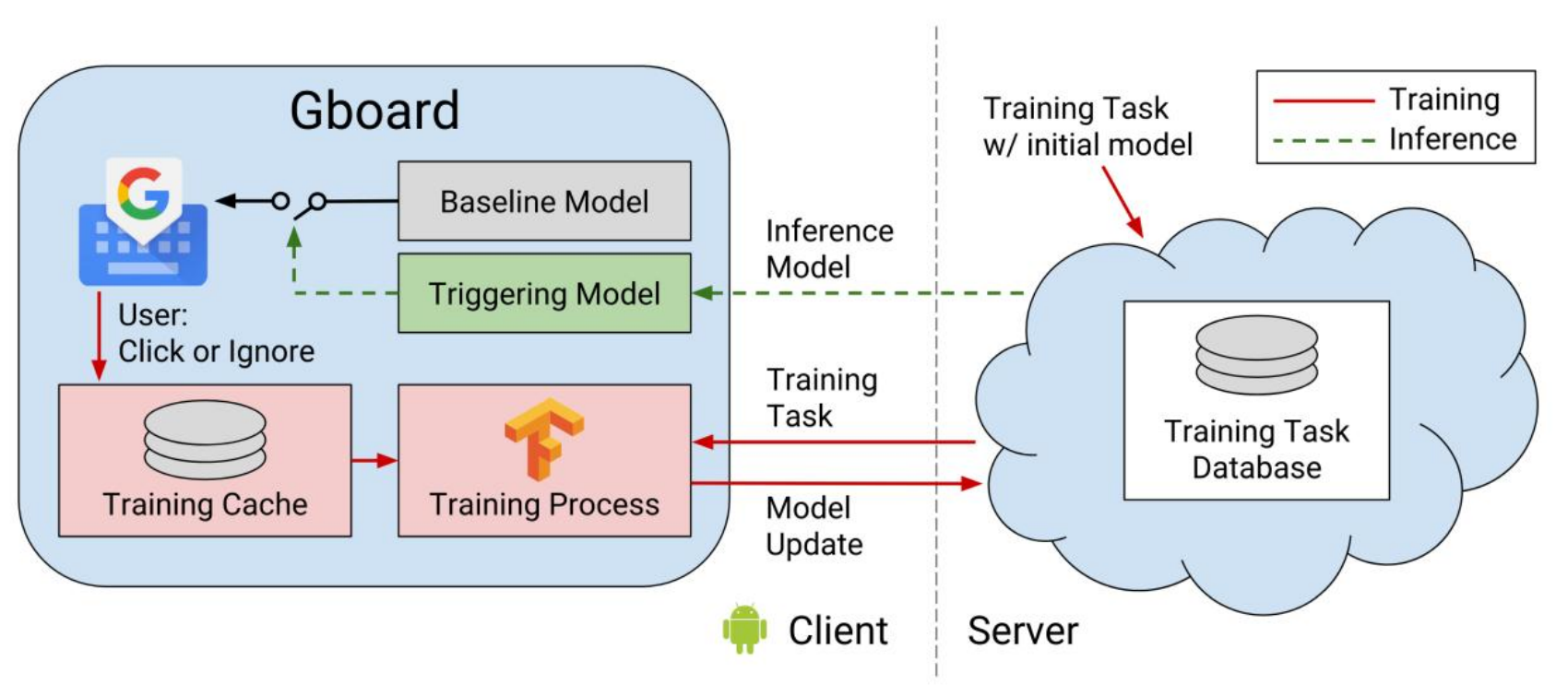
Application

- Gboard
- Recommender system
- Autonomous driving
- …

Application

Gboard

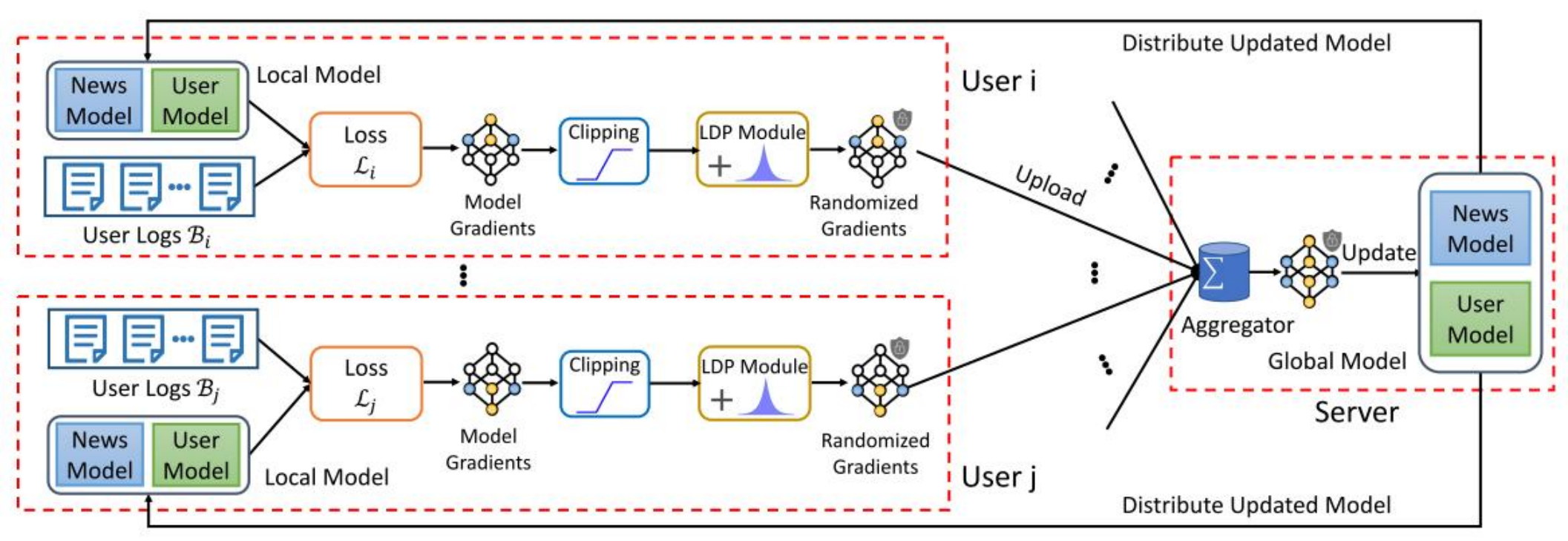
- Google's first implementation of federated learning.
- Triggering model is trained federated to tune the results of the pre-trained baseline model for better performance.



Application

Recommender system

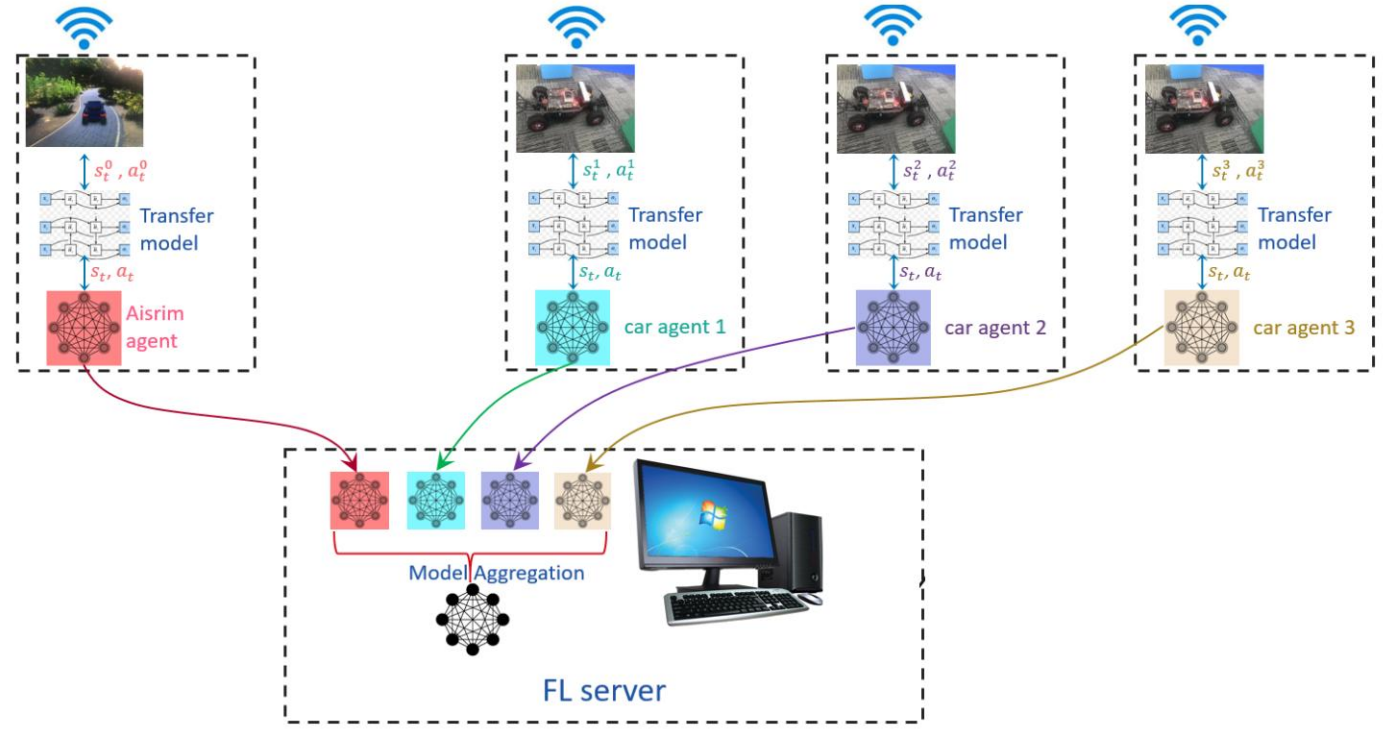
- The news model aims to learn news representations to model news content.
- The user model is used to learn user representations to model their personal interest.
- LDP denotes the local differential privacy



Application

Autonomous driving

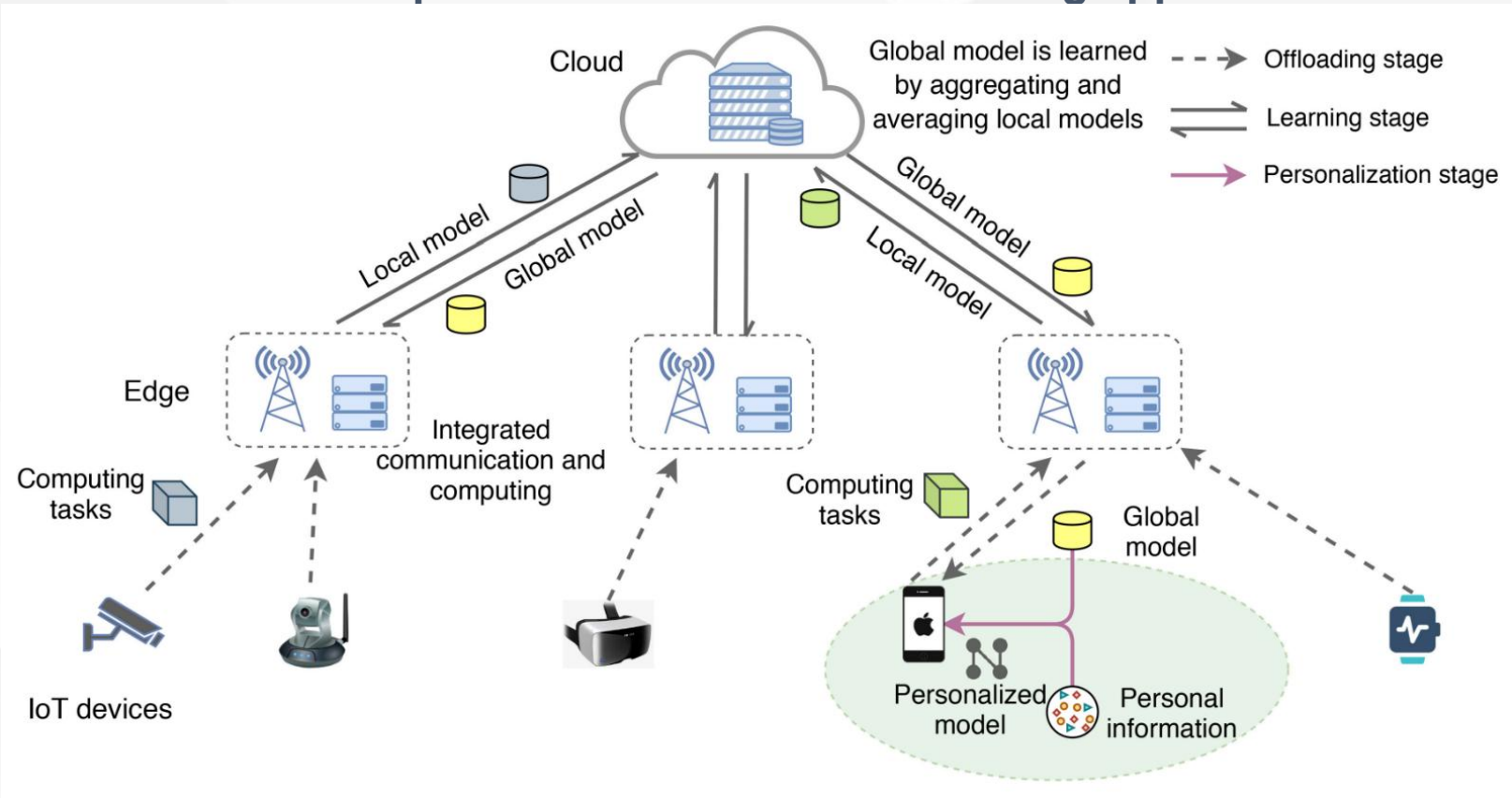
- The FTRL framework for collision avoidance RL tasks of autonomous driving cars
- Global model is asynchronously updated by different RL agents.
- Transfer knowledge from virtual world (Airsim platform) to real world



Application

IOT

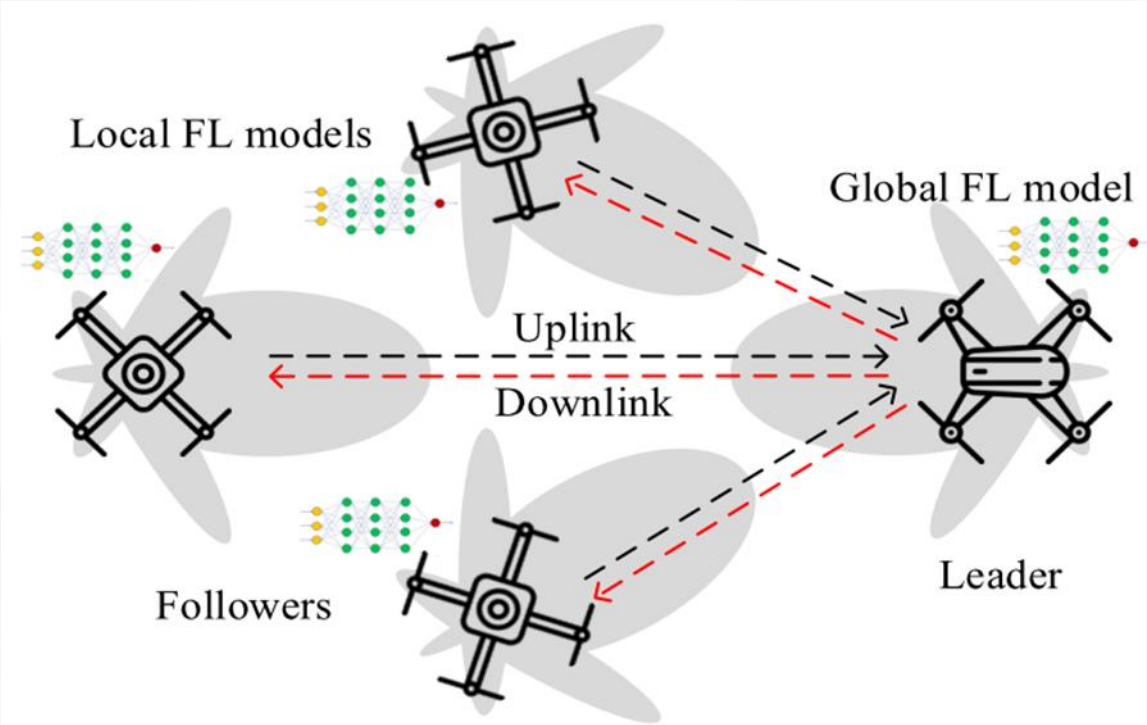
- Personalized federated learning framework for intelligent IoT applications.
- Supports flexible selection of personalized federated learning approaches.



Application

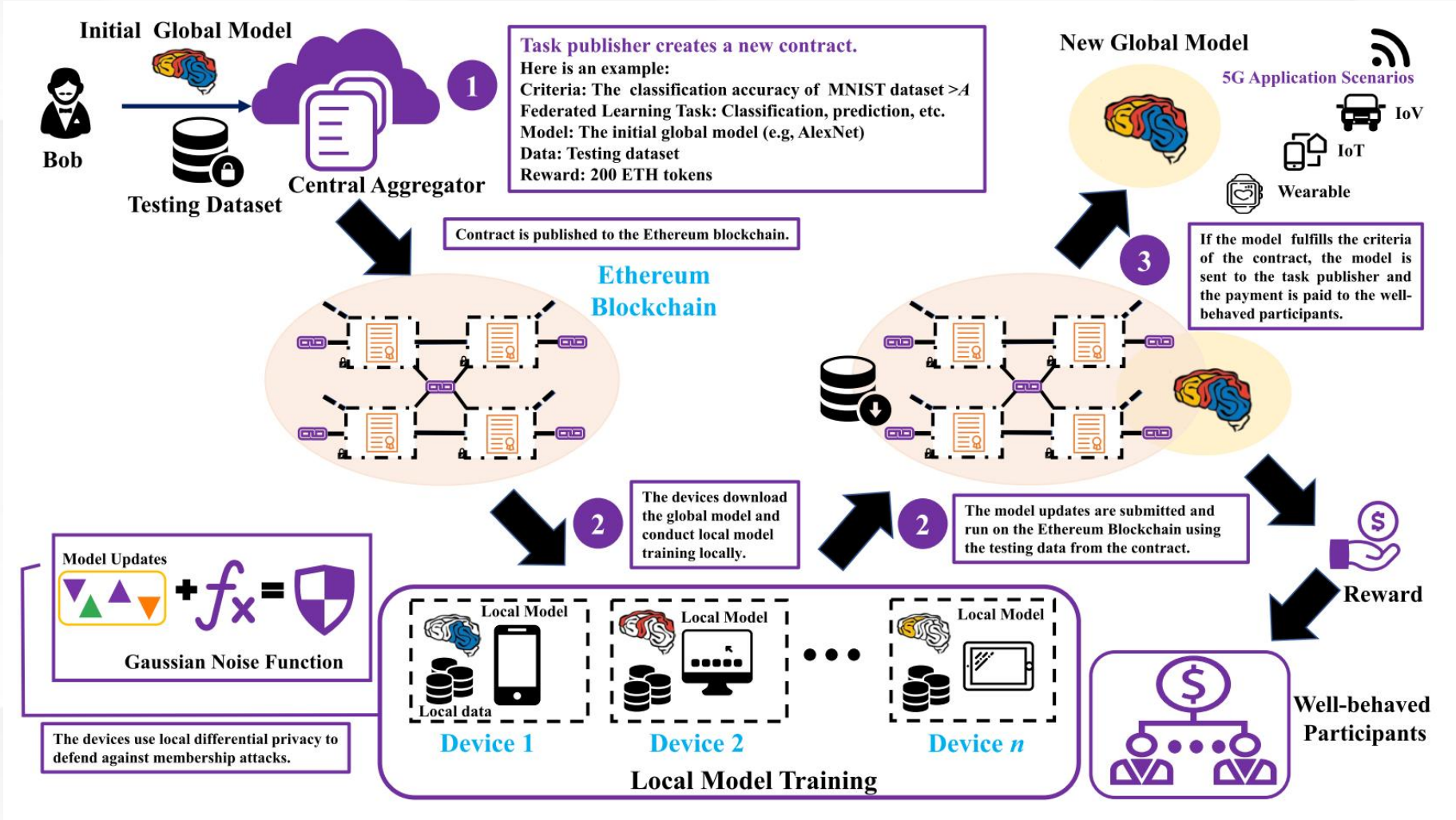
UAV (Unmanned aerial vehicle)

- Due to the high mobility of UAVs and their limited energy and stringent energy limitations, the analysis in previous federated learning work cannot be directly applied for UAV swarms.
- Use a sample average approximation approach from stochastic programming along with a dual method from convex optimization.



Application

Blockchain



Section 2: Frontier Research of Serverless Computing

Embrace Cloud Computing

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录

01

NumPyWren

- Serverless Linear Algebra

NumPyWren Overview

Background

- Current distributed programming abstractions such as MPI and MapReduce rely on the tightly integrated resources in a collection of individual servers.
- To write applications for a disaggrated datacenter, the datacenter operator must expose a new programming abstraction.

Motivation

- Serverless computing is a programming model in which the cloud provider manages the servers, and also dynamically manages the allocation of resources.
- Disaggregation can provide benefits to linear algebra tasks as these workloads have large dynamic range in memory and computation requirements.

Contribution

- large scale linear algebra algorithms can be efficiently executed using **stateless functions and disaggregated storage**
- design LAmbdaPACK, a **domain specific language for linear algebra algorithms**
- NumPyWren can **scale** to run Cholesky decomposition

Background: Serverless Computing

Cloud providers offer the ability to execute functions on demand, hiding cluster configuration and management overheads from end users.

① Cloud providers offer a number of **storage** options ranging from key-value stores to relational databases.

- The cost of data storage in an object storage system is often orders of magnitude lower when compared to instance memory.

② Cloud providers also offer **publish-subscribe services** like Amazon SQS or Google Task Queue.

Background: Serverless Computing

Cloud providers offer the ability to execute functions on demand, hiding cluster configuration and management overheads from end users.

③ **Computation resources** offered in serverless platforms are typically restricted to a single CPU core and a short window of computation.

- AWS Lambda provides 900 seconds of compute on a single AVX core with access to up to 3 GB of memory and 512 MB of disk storage.

④ The linear scalability in function execution is only useful for embarrassingly parallel computations when there is no **communication** between the individual workers.

Background: Linear Algebra Algorithms

- **Cholesky factorization** is one of the most popular algorithms for solving linear equations, and it is widely used in applications such as matrix inversion, partial differential equations, and Monte Carlo simulations.

$Ax = b$	
$A = LL^T$	$O(n^3)$
$Ly = b$	$O(n^2)$
$L^T x = y$	$O(n^2)$

Communication-Avoiding Cholesky

Algorithm 1 Communication-Avoiding Cholesky [5]

Input:

A - Positive Semidefinite Symmetric Matrix

B - block size

N - number of rows in A

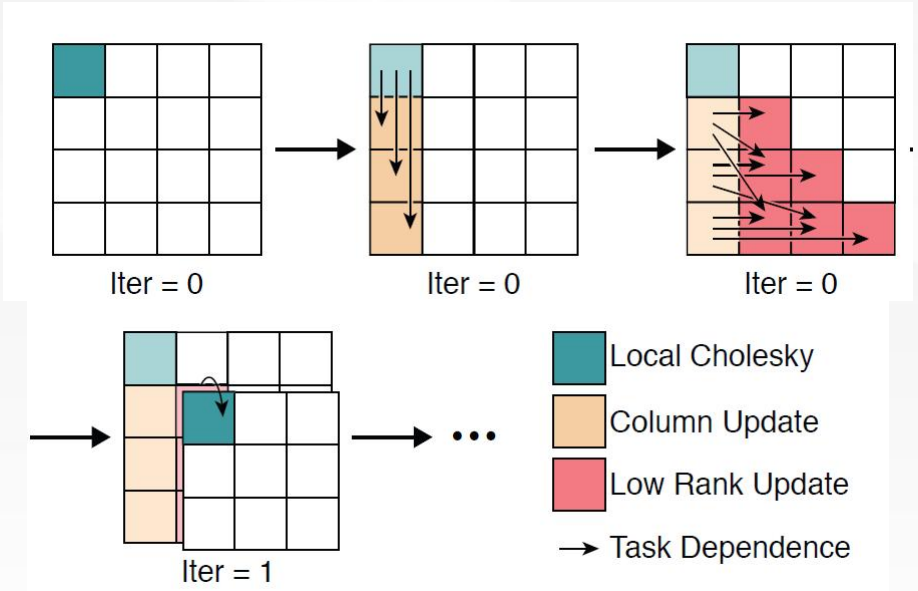
Blocking:

A_{ij} - the ij-th block of A

Output:

L - Cholesky Decomposition of A

1: **for** $j \in \{0 \ldots \lceil \frac{N}{B} \rceil\}$ **do**
2: $L_{jj} \Leftarrow cholesky(A_{jj})$
3: **for all** $i \in \{j+1 \ldots \lceil \frac{N}{B} \rceil\}$ **do in parallel**
4: $L_{ij} \Leftarrow L_{jj}^{-1} A_{ij}$
5: **end for**
6: **for all** $k \in \{j+1 \ldots \lceil \frac{N}{B} \rceil\}$ **do in parallel**
7: **for all** $l \in \{k \ldots \lceil \frac{N}{B} \rceil\}$ **do in parallel**
8: $A_{kl} \Leftarrow A_{kl} - L_{kj}^{T} L_{lj}$
9: **end for**
10: **end for**
11: **end for**



① Diagonal block Cholesky decomposition

② Parallel column update

③ Parallel submatrix update

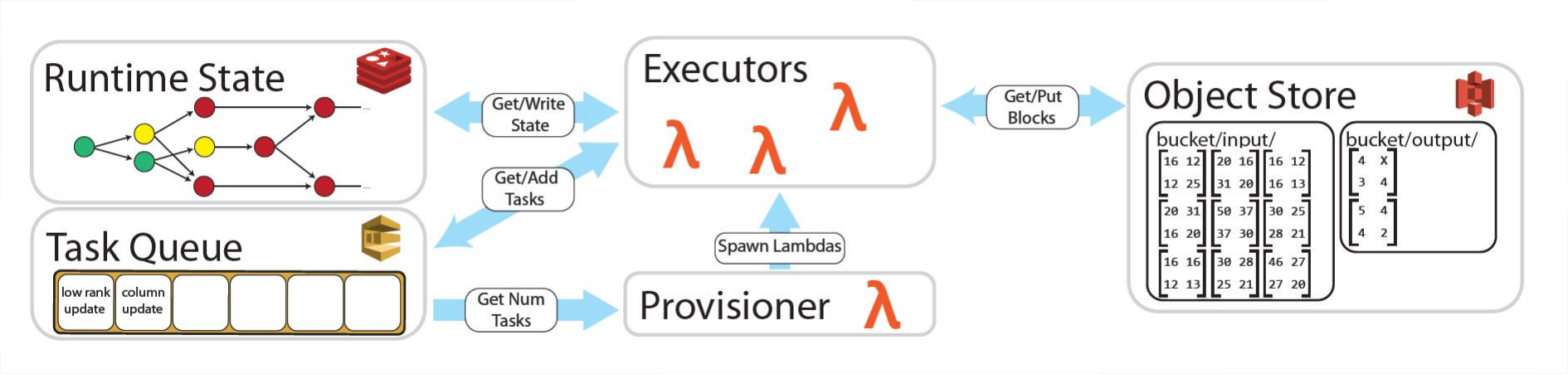
④ Diagonal block Cholesky decomposition

❶ dynamic parallelism

❷ fine-grained dependencies

System Design

- **Task Enqueue:** enqueue the first task that needs to be executed into the task queue
- **Executor Provisioning:** launch an executor, and maintain the number of active executors based on task queue size
- **Task Execution:** manage executing and scheduling NumPyWren tasks
- **Runtime State Update:** update the task status in the runtime state store



The architecture of the execution framework of NumPyWren showing the runtime state during a 6x6 Cholesky decomposition. The first block Cholesky instruction has been executed as well as a single column update.

System Design

Fault tolerance in NumPyWren is much simpler to achieve due to the disaggregation of compute and storage.

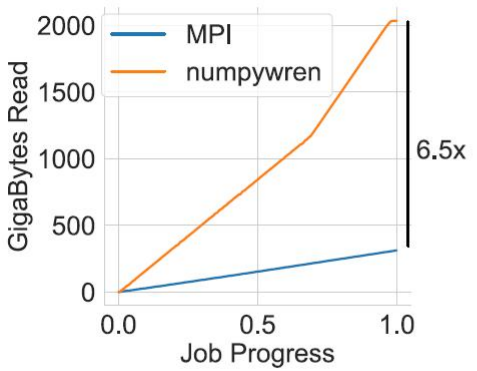
- **Task Lease:** NumPyWren executes failed tasks via a lease mechanism, which allows the system to track task status without a scheduler periodically communicating with executors.
- **Failure Detection and Recovery:** Failure detection happens through lease expiration and recovery latency is determined by lease length.
- **Garbage Collection:** it is imperative we clear the state when it is no longer necessary.
- **Autoscaling**
  - Task scheduling and worker management is decoupled in NumPyWren, which allows auto-scaling of computing resources for a better cost-performance trade-off.
  - We adopt a simple auto-scaling heuristic and it achieves good utilization while keeping job completion time low.

Evaluation

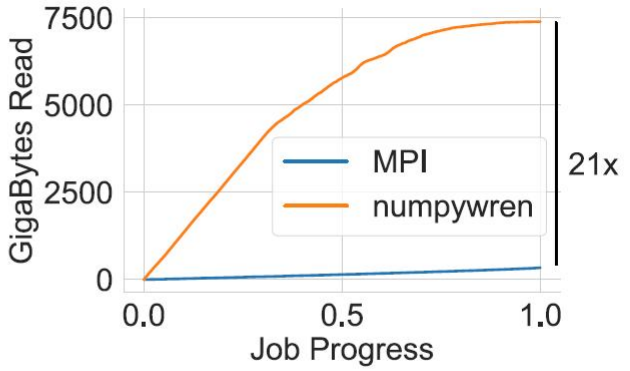
System Comparisons

- The amount of bytes read by NumPyWren is always greater than MPI.
- Even though NumPyWren reads more than 21x bytes over the network when compared to MPI, our end to end completion time is only 47% slower.

Algorithm	MPI (sec)	NumPyWren (sec)	Slow down
SVD	5.8e4	4.8e4	N/A
QR	9.9e3	1.4e4	1.5x
GEMM	5.0e3	8.1e3	1.6x
Cholesky	1.7e3	2.5e3	1.5x



(a) GEMM



(b) QR

Evaluation

System Comparisons

- For MPI the core-seconds is the total amount of cores multiplied by the wall clock runtime.
- For NumPyWren we wish to only account for “active cores" in our core-second calculation, as the free cores can be utilized by other tasks.
- NumPyWren can achieve resource savings of over 3x for the SVD algorithm.

Algorithm	MPI (core-secs)	NumPyWren (core-secs)	Resource saving
SVD	2.1e7	6.2e6	3.4x
QR	2.6e6	2.2e6	1.15x
GEMM	1.2e6	1.9e6	0.63x
Cholesky	4.5e5	3.9e5	1.14x

Evaluation

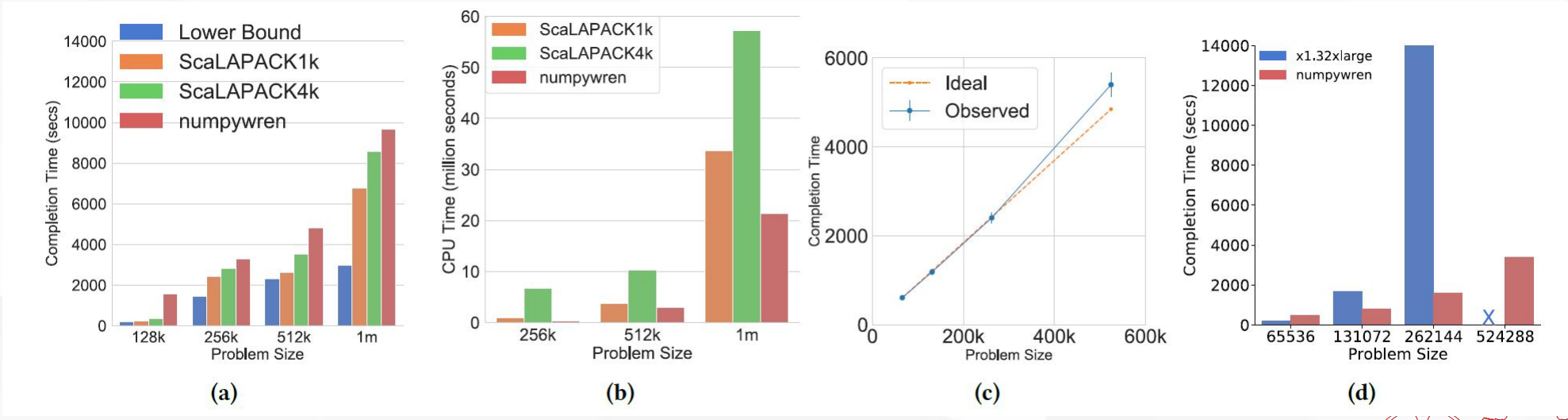
Scalability

a) Completion time on various problem sizes when NumPyWren is run on same setup as ScaLAPACK

b) Total execution core-seconds for Cholesky when the NumPyWren and ScaLAPACK are optimized for utilization.

c) Weak scaling behavior of NumPyWren.

d) Comparison of NumPyWren with 128 core single node machine running Cholesky decompositions of various sizes



02

Cirrus

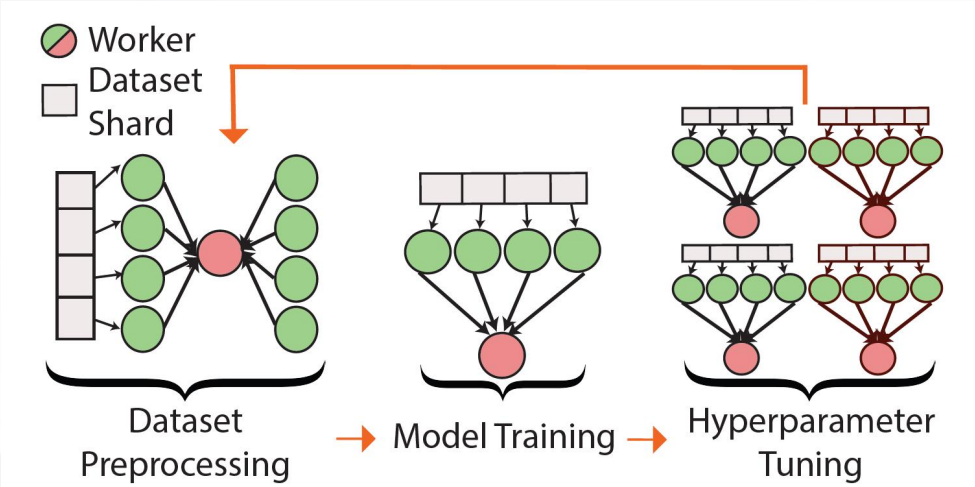
- Cirrus: a Serverless Framework for End-to-end ML Workflows

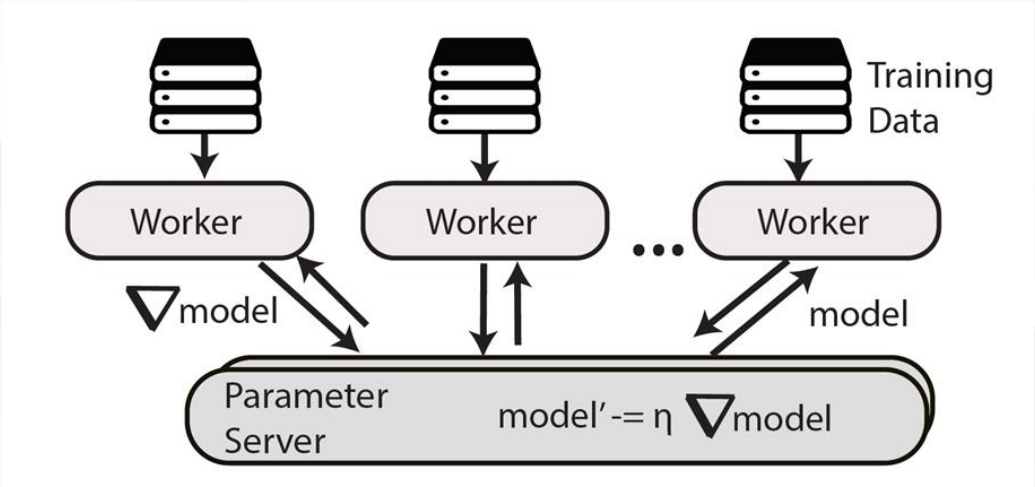
Cirrus Overview

- Background
  - The widespread adoption of ML techniques in a wide-range of domains has made machine learning one of the leading revenue-generating datacenter workloads.
  - The complexity of ML workflows leads to two problems, over-provisioning and explicit resource management.
- Motivation
  - Serverless computing relies on the cloud infrastructure to automatically address the challenges of resource provisioning and management.
  - The benefits of serverless computing for ML hinge on the ability to run ML algorithms efficiently.
- Contribution
  - Cirrus is designed to efficiently support the entire ML workflow.
  - Cirrus builds on three key design properties, ultra-lightweight, cost-saving, and stateless.
  - It yields a 3.75x improvement on time-to-accuracy compared to the best-performing

Background: End-to-end ML Workflow

- **Dataset preprocessing** typically involves an expensive map/reduce operation on data.
- **Model training**: Workers consume data shards, compute gradients, and synchronize with a parameter server.
- **Hyperparameter optimization** to tune model and training parameters involves running multiple training instances.





Background: Challenges

Machine Learning

- **Over-provisioning:** The heterogeneity of the different tasks in an ML workflow leads to a significant resource imbalance during the execution of a training workflow.
- **Explicit resource management:** Systems that leverage VMs for machine learning workloads generally require users to repeatedly perform a series of onerous tasks.

Serverless Computing

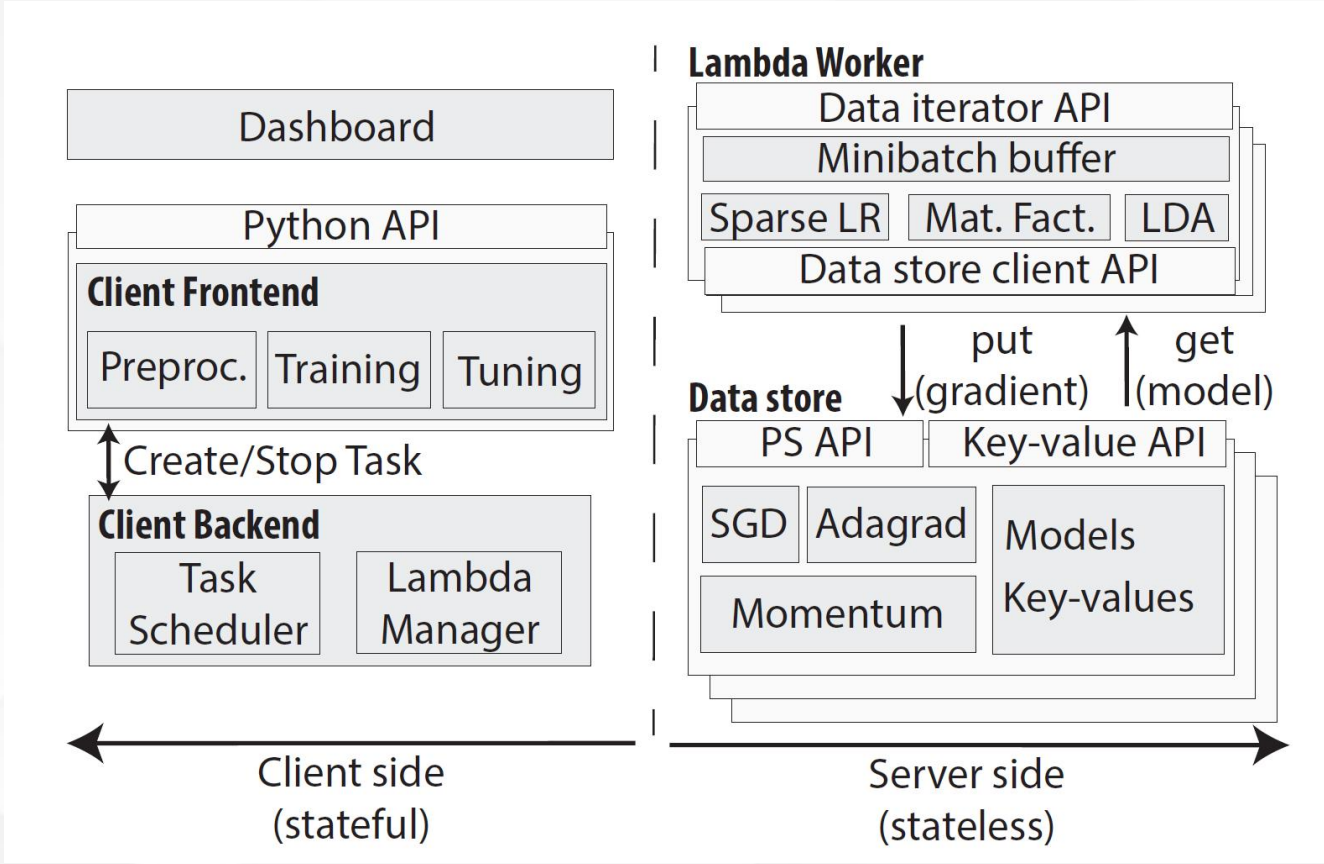
- **Small local memory and storage:** Lambda functions, by design, have very limited memory and local storage.
- **Low bandwidth and lack of P2P communication:** Lambda functions have limited available bandwidth when compared with a regular VM.
- **Short-lived and unpredictable launch times:** Lambda functions are short-lived and their launch times are highly variable.
- **Lack of fast shared storage:** Because lambda functions cannot connect between themselves, shared storage needs to be used.

Design: Principles

- **Adaptive, fine-grained resource allocation**
  - To avoid resource waste that arises from over-provisioning, Cirrus should flexibly adapt the amount of resources reserved for each workflow phase with fine-granularity.
- **Stateless server-side backend**
  - To ensure robust and efficient management of serverless compute resources, Cirrus, by design, operates a stateless, server-side backend.
- End-to-end serverless API
  - Model training is not the only important task an ML researcher has to perform.
- High scalability
  - ML tasks are highly compute intensive, and thus can take a long time to complete without efficient parallelization.

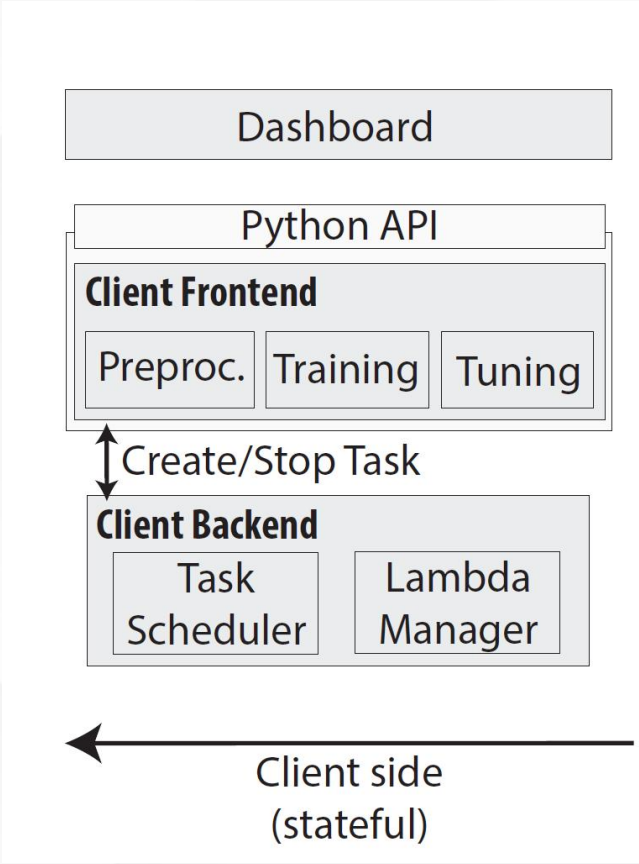
Design: Framework

- **Client Side**
  - Client Frontend
  - Client Backend
- **Server Side**
  - Lambda Worker
  - Data Store



Design: Client Side

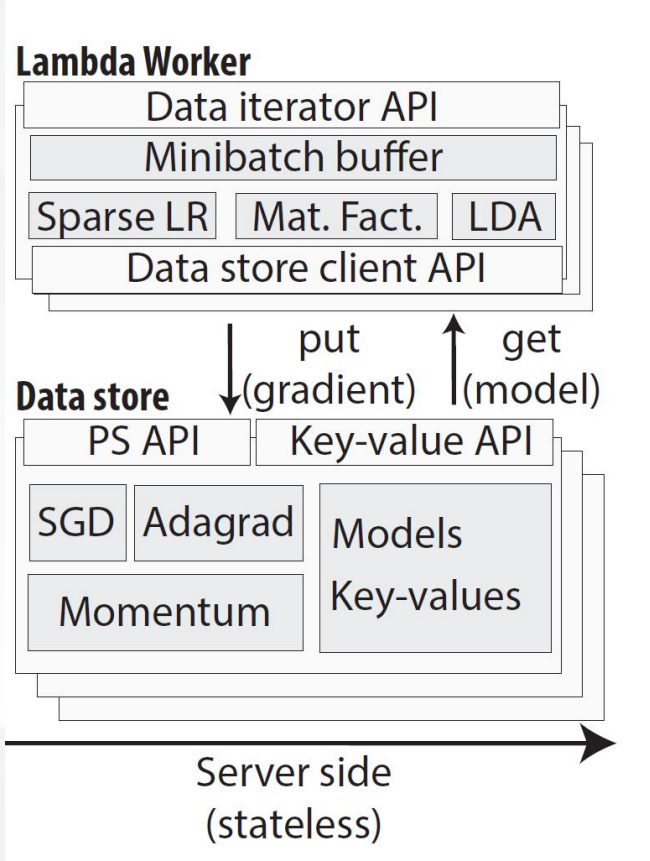
- Python frontend
  - Preprocessing
  - Training
  - Hyperparameter optimization
- Client-side backend
  - parse training data and load it to S3
  - launch the Cirrus workers on lambdas
  - manage the distributed data store
  - keep track of the progress of computations
  - return results to the Python frontend



Design: Server Side

- Worker runtime
  - a smart iterator for training datasets stored in S3
  - provides an API for the distributed data store
- Distributed data store

API	Description
int send_gradient_X(ModelGradient* g)	Sends model gradient
SparseModel get_sparse_model_X(const std::vector<int>& indices)	Get subset of model
Model get_full_model_X()	Get all model weights
set_value(string key, char* data, int size)	Set intermediate state
std::string get_value(string key)	Get intermediate state



Design: End-to-end Workflow

```python
import cirrus
import numpy as np

local_path = "local_criteo"
s3_input = "criteo_dataset"
s3_output = "criteo_norm"

cirrus.load_libsvm(local_path, s3_input)

cirrus.normalize(s3_input, s3_output,
                 MIN_MAX_SCALING)
```

(a) Pre-process

```python
params = {
 'n_workers': 5,
 'n_ps': 1,
 'worker_size': 1024,
 'dataset': s3_output,
 'epsilon': 0.0001,
 'timeout': 20 * 60,
 'model_size': 2**19,
}

lr_task =  cirrus.LogisticRegression(params)
result = lr_task.run()
```

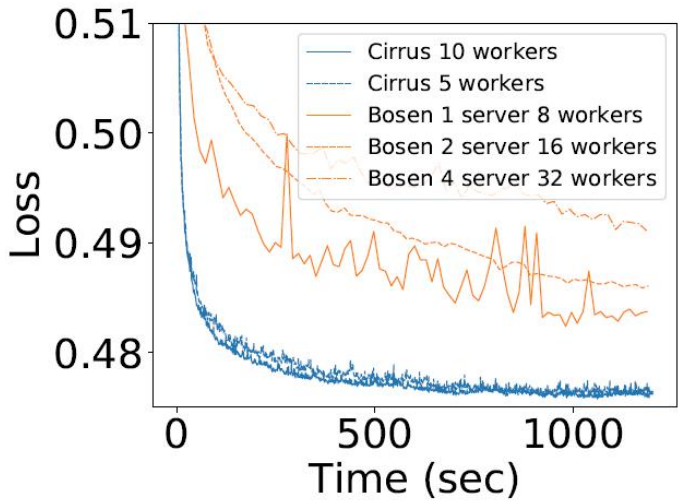
(b) Train

```python
# learning rates
lrates = np.arange(0.1, 10, 0.1)
minibatch_size = [100, 1000]

gs = cirrus.GridSearch(
 task=cirrus.LRegression,
 param_base=params,
 hyper_vars=["learning_rate", "minibach_size"],
 hyper_params=[lrates, minibatch_size])

results = gs.run()
```

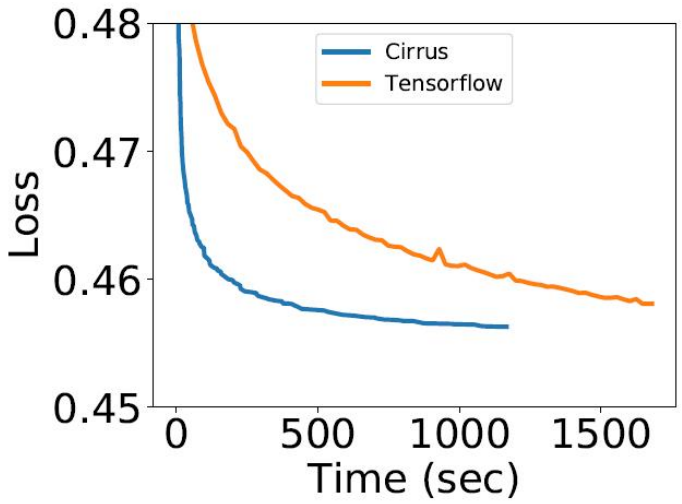
(c) Tune

Evaluation: Sparse Logistic Regression

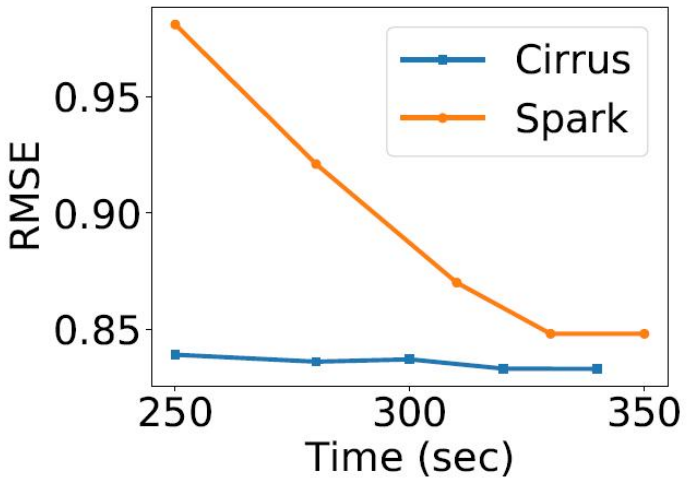
- Baseline
  - Bosen
  - TensforFlow
  - Spark



(a) Bosen



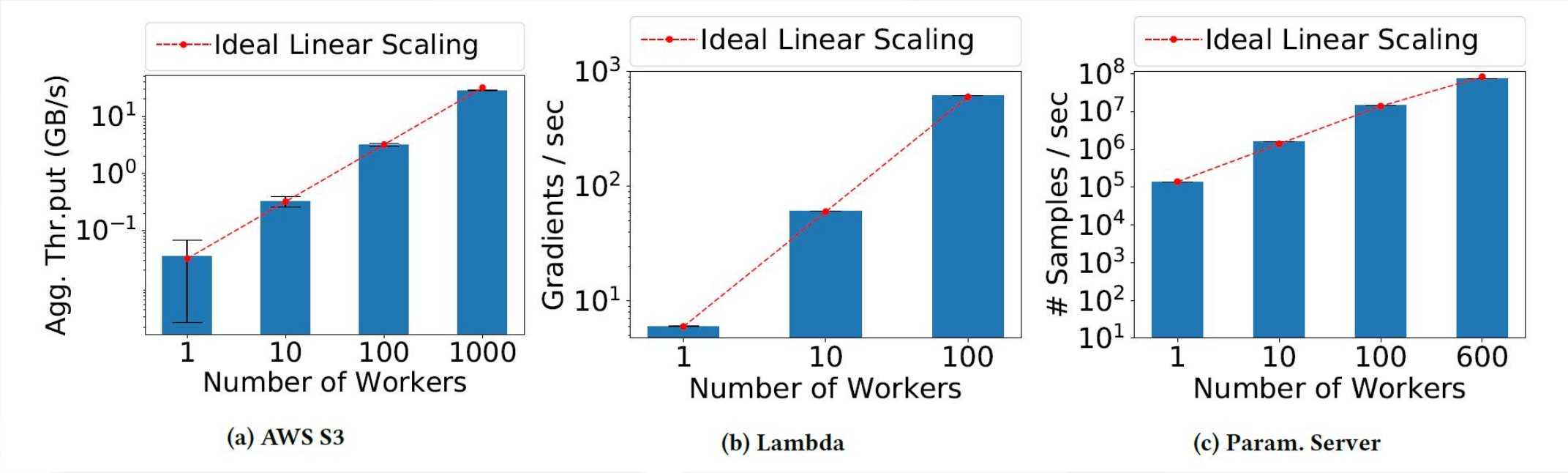
(b) Tensorflow



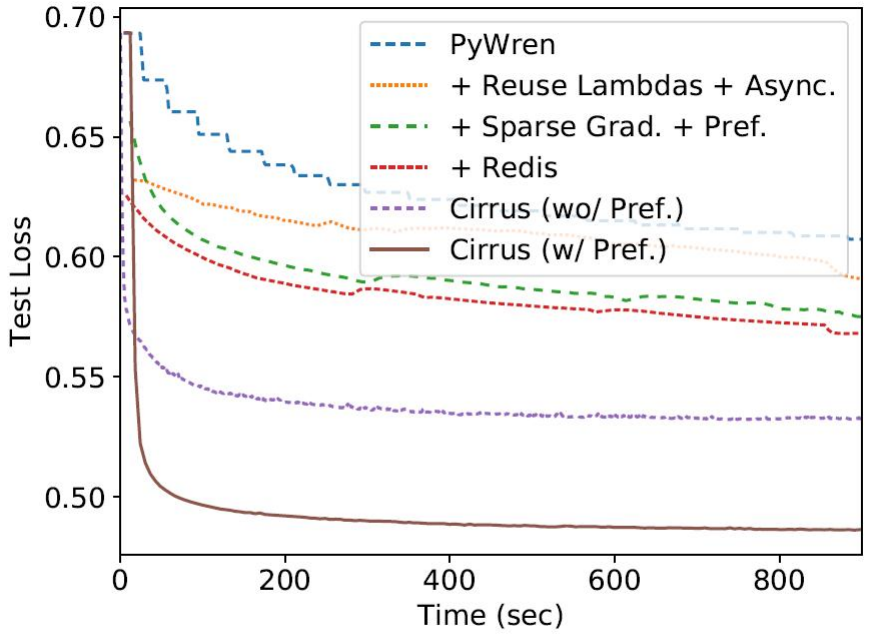
(c) Spark

Evaluation: Scalability

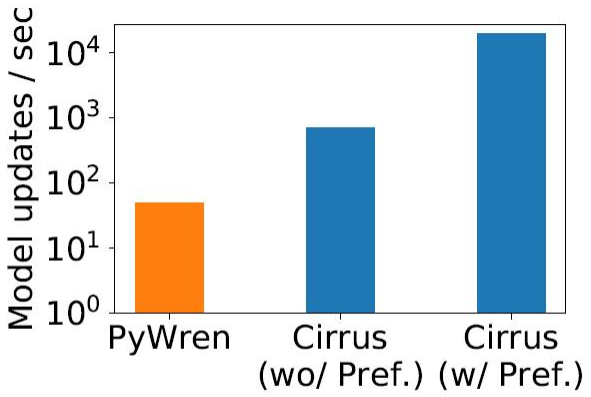
- Storage scalability
- Compute scalability
- Parameter server scalability



Evaluation: The Benefits of ML Specialization



(a) Convergence over time.



(b) Model updates per second.

03

LambdaML

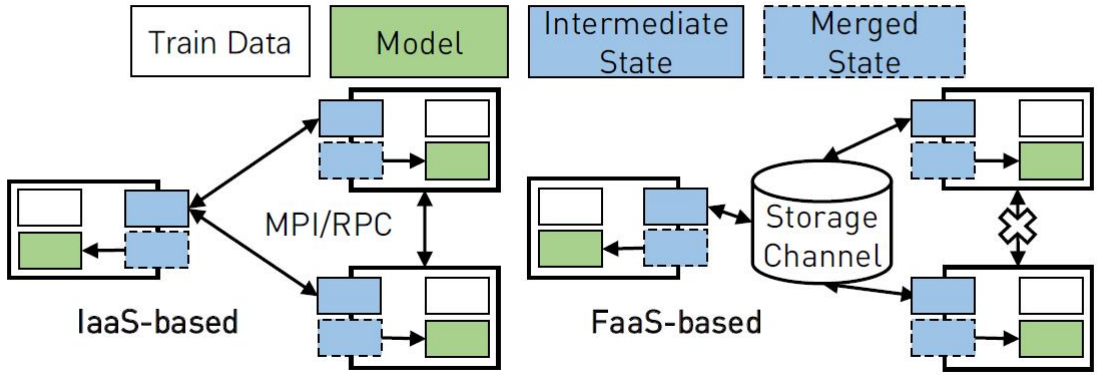
- Towards Demystifying Serverless Machine Learning Training

Background: Distributed Machine Learning

- Data and Model
- Optimization Algorithm
  - In each iteration, the training procedure would typically scan the training data, compute necessary quantities (e.g., gradients), and update the model.
  - Training ML models in a distributed setting is more complex, due to the extra complexity of distributed computation as well as coordination of the communication between executors.
- Communication Mechanism
  - **Communication Channel:** The efficiency of data transmission relies on the underlying communication channel.
  - **Communication Pattern:** Gather, AllReduce, and ScatterReduce
  - **Synchronization Protocol:** bulk synchronous parallel (BSP), asynchronous parallel (ASP)

Background: FaaS vs. IaaS for ML

- IaaS: users have to build a cluster by renting VMs or reserve a cluster with predetermined configuration parameters
  - Cons: There is no elasticity or auto-scaling if the reserved computation resources turn out to be insufficient.
- FaaS
  - Pros: Resource allocation in FaaS is on-demand and auto-scaled, and users are only charged by their actual resource usages.
  - Cons: FaaS currer ling strategies.



Design: System Overview

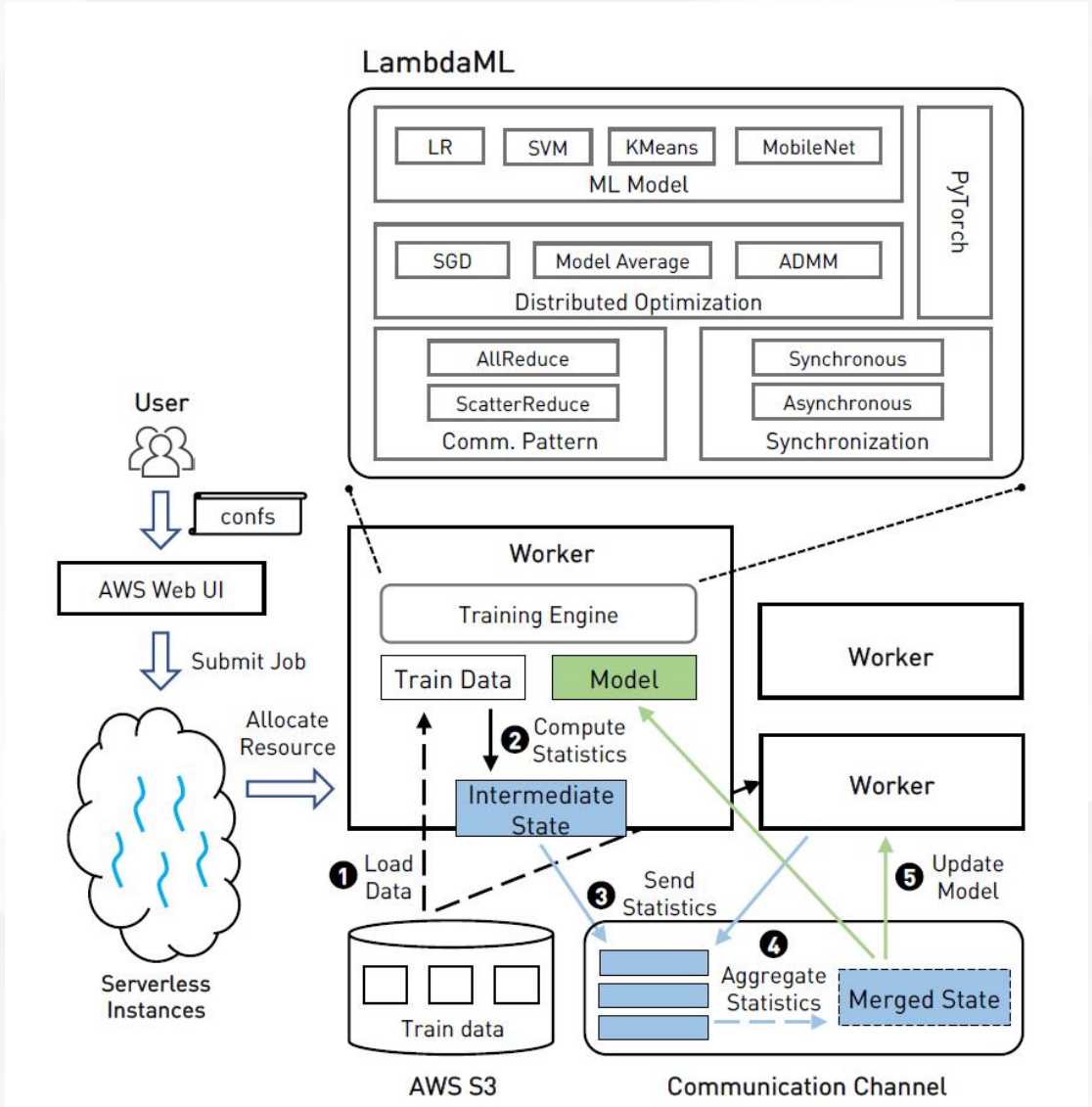
① Load data

② Compute statistics

③ Send statistics

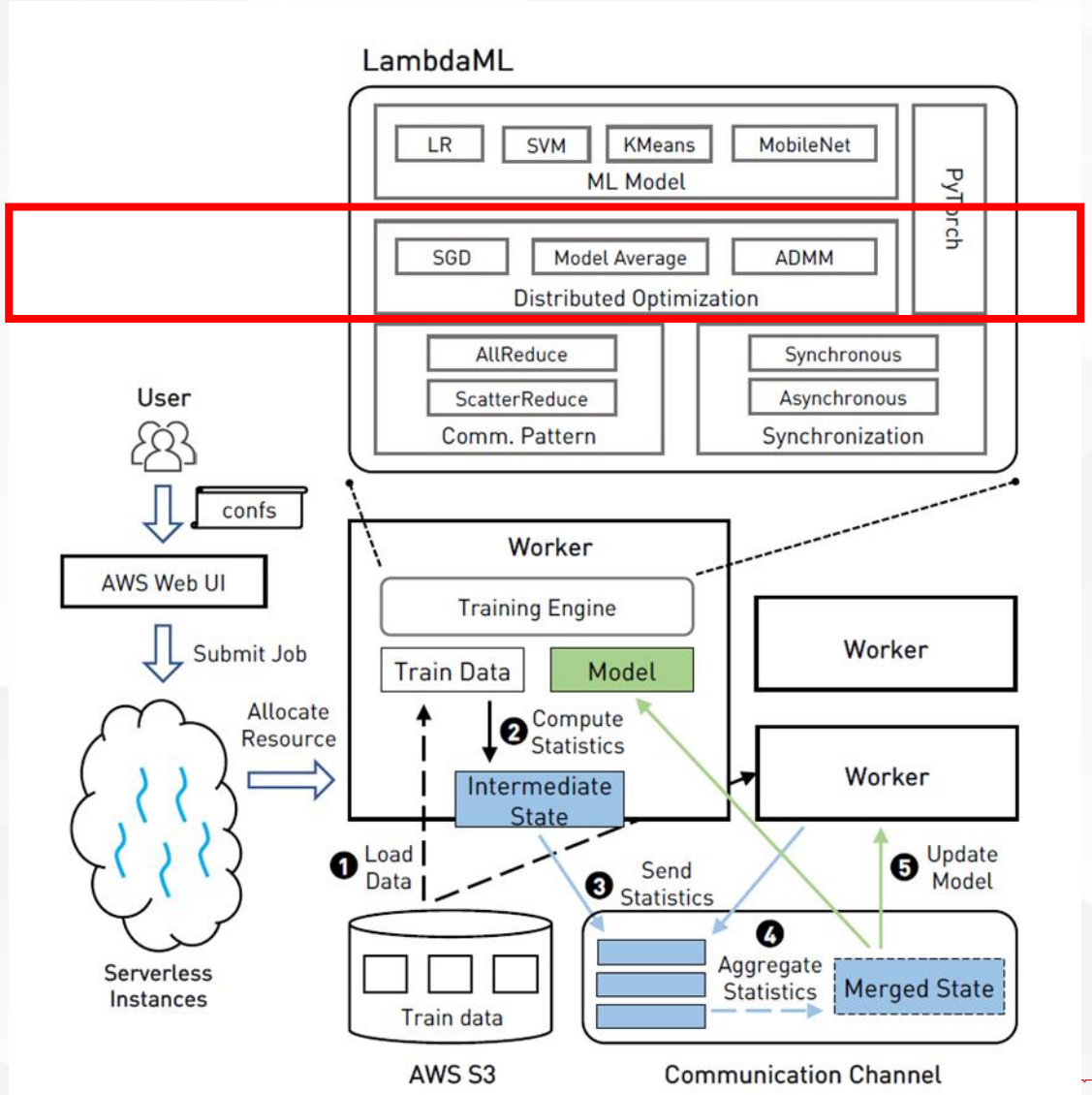
④ Aggregate statistics

⑤ Update model



Design: Distributed Optimization Algorithm

- Distributed SGD
  - Stochastic gradient descent (SGD) is perhaps the most popular optimization algorithm.
  - **Gradient Averaging:** GA updates the global model in every iteration by harvesting and aggregating the (updated) gradients from the executors.
  - **Model Averaging:** MA collects and aggregates the (updated) local models.
- Distributed ADMM
  - ADMM breaks a large-scale convex optimization problem into several smaller subproblems



Design: Communication Channel

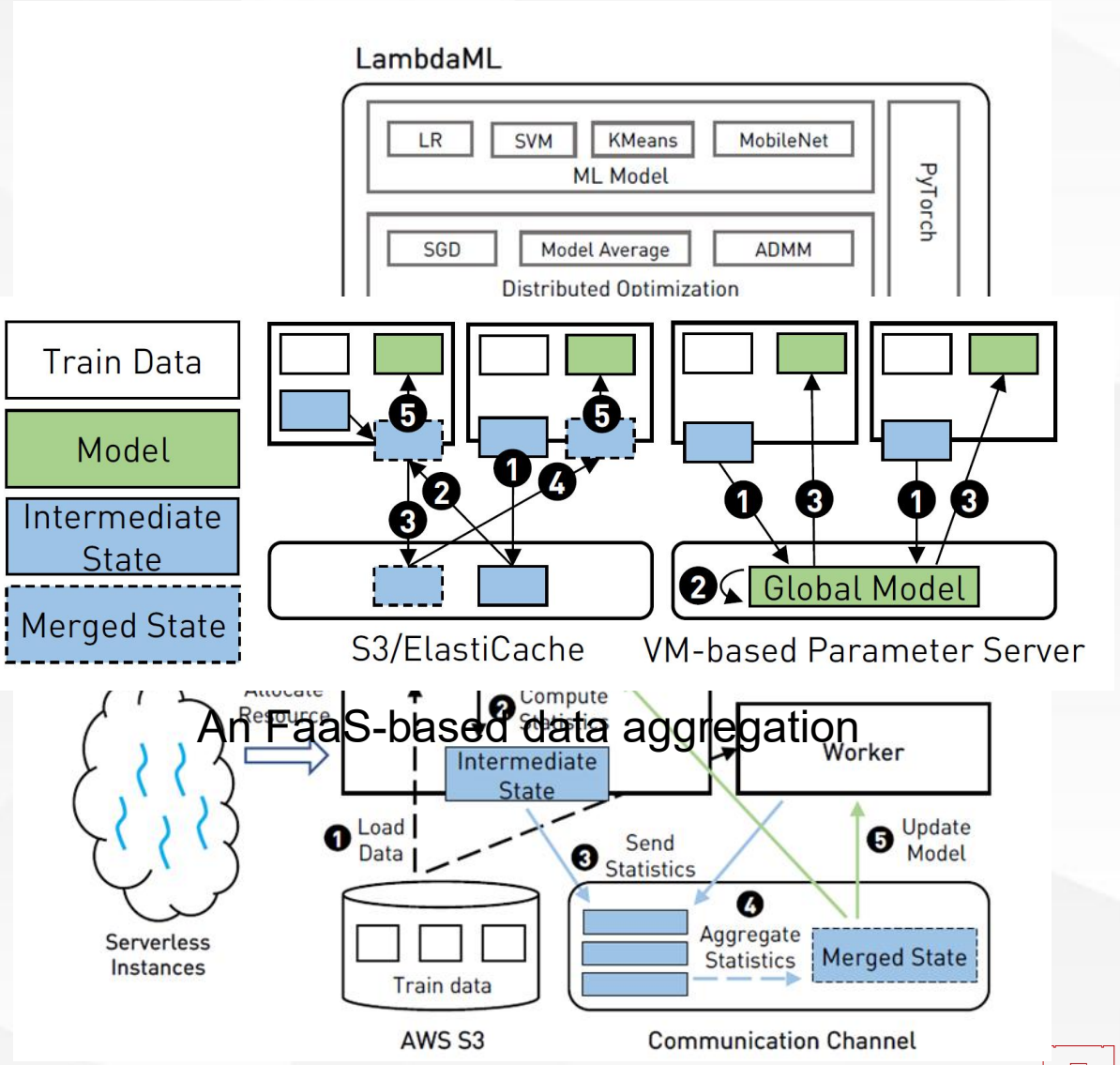
① Each executor stores its generated intermediate data as a temporary file in S3;

② The first executor pulls all temporary files from the storage service and merges them to a single file;

③ The leader writes the merged file back to the storage service;

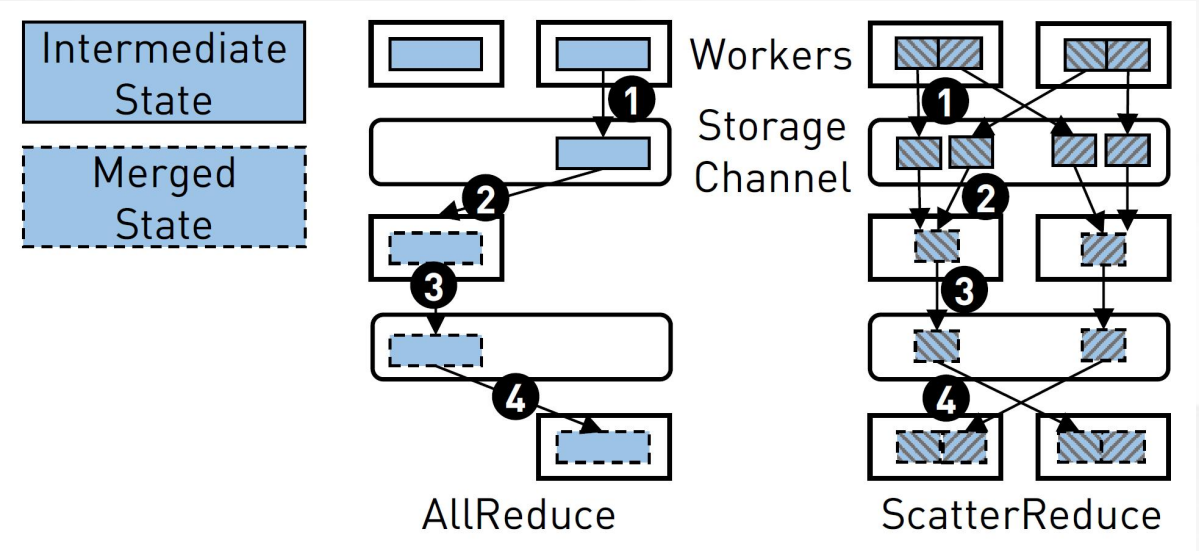
④ All the other executors (except the leader) read the merged file from the storage service;

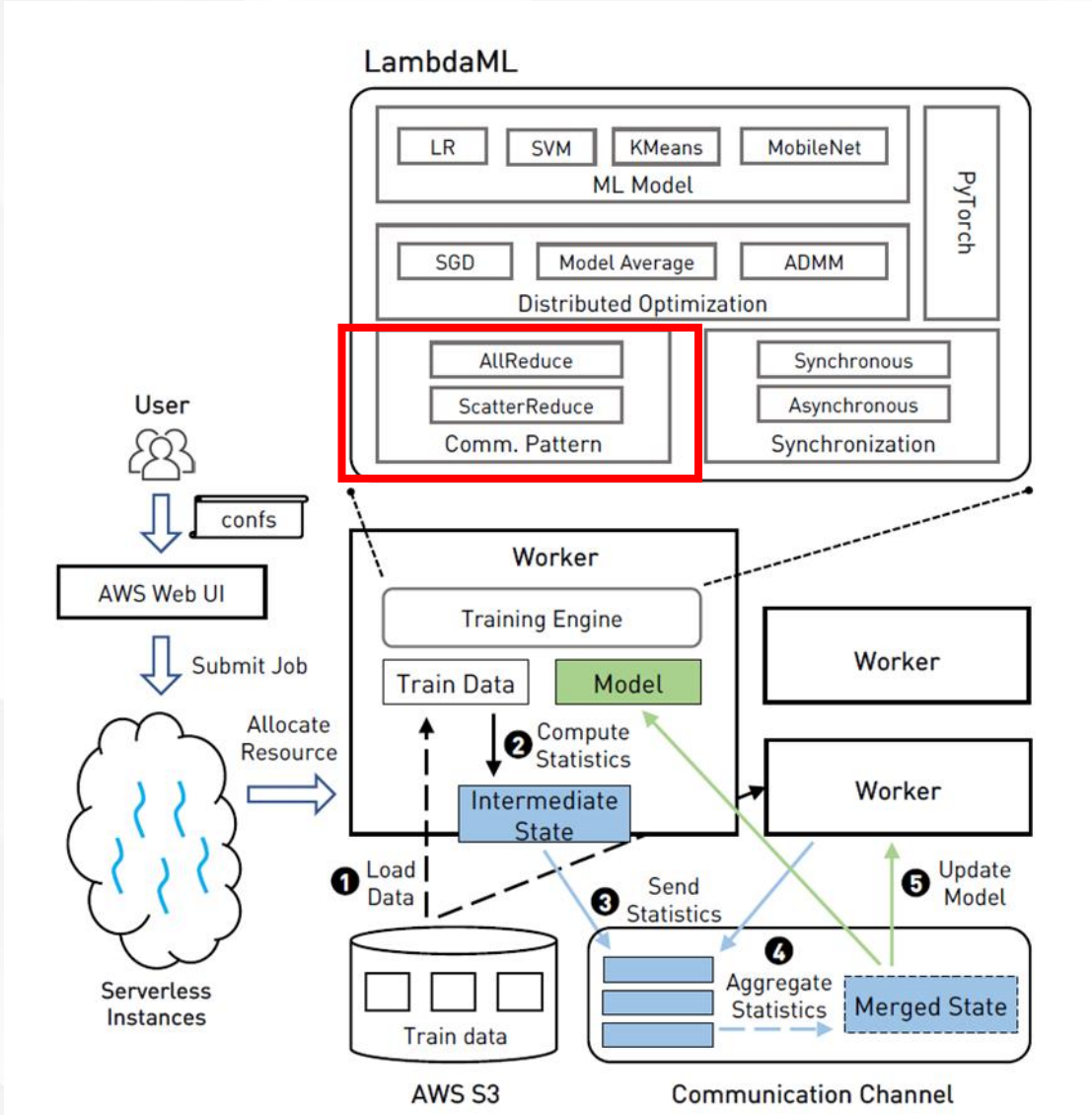
⑤ All executors refresh their (local) model with information read from the merged file.



Design: Communication Pattern

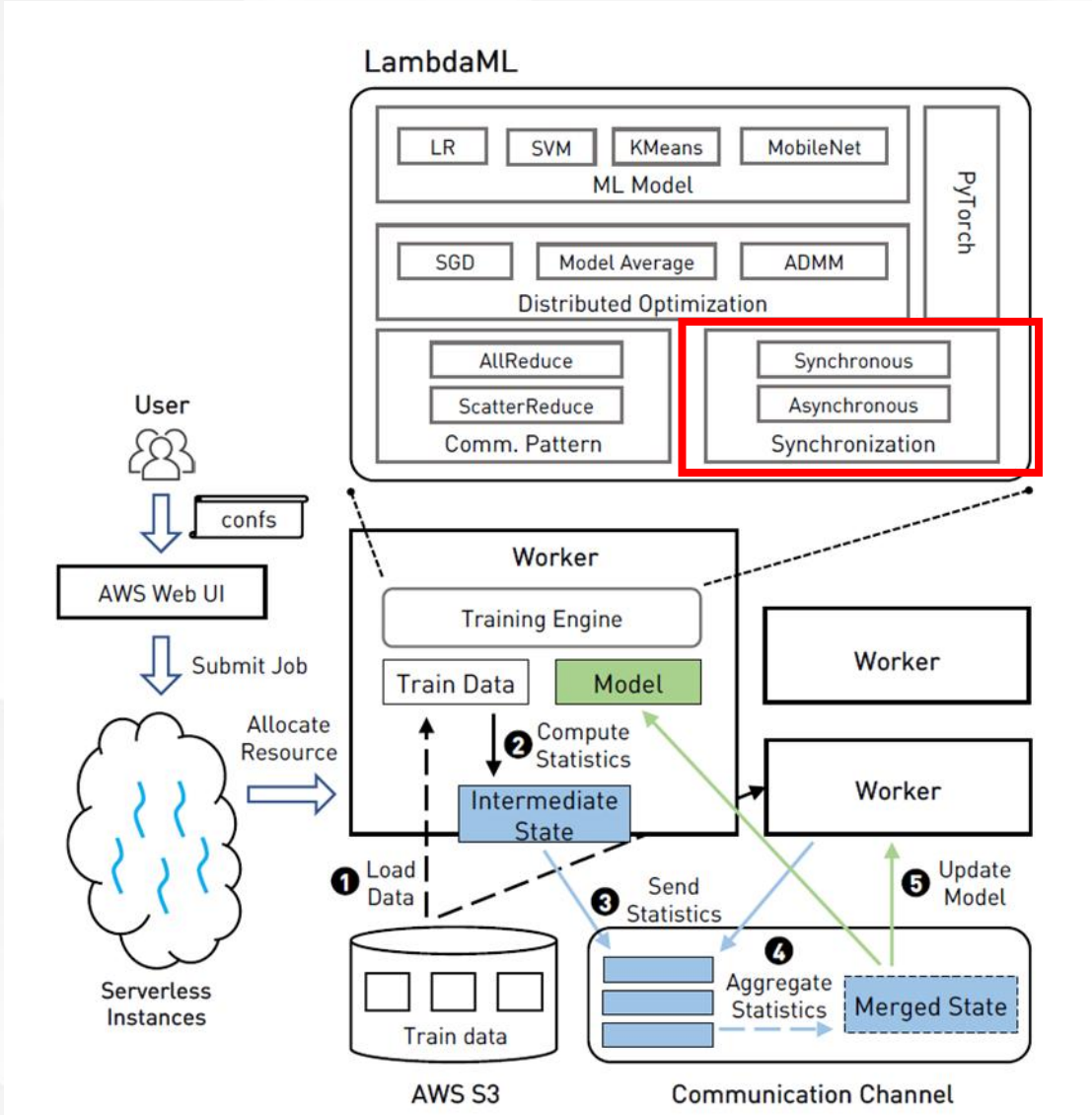
- AllReduce
- ScatterReduce



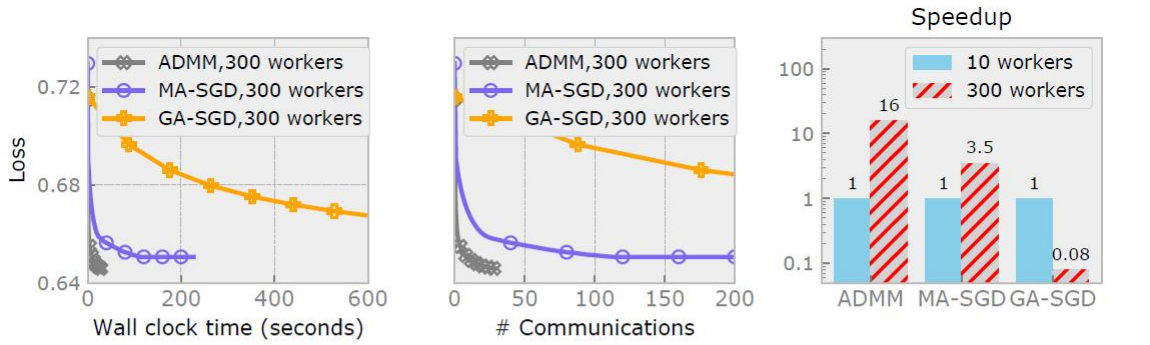


Design: Synchronization Protocol

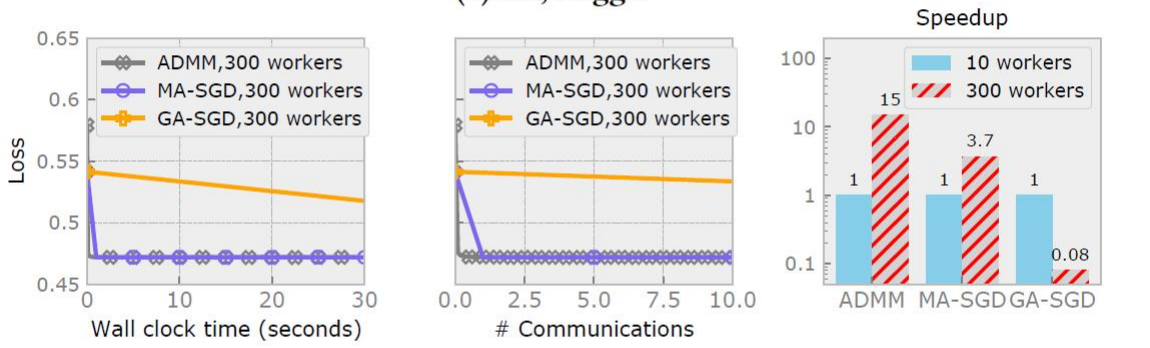
- Synchronous
  - Merging phase: All executors first write their local updates to the storage service. The reducer/aggregator waits all the other executors.
  - Updating phase: The aggregator finishes aggregating all data and stores the aggregated information back to the storage service.
- Asynchronous
  - One replica of the trained model is stored on the storage service as a global state.
  - Each executor runs independently – it reads the model from the storage service, updates the model with training data, writes
  - the new model back to the storage service – without caring about the speeds of the other executors.



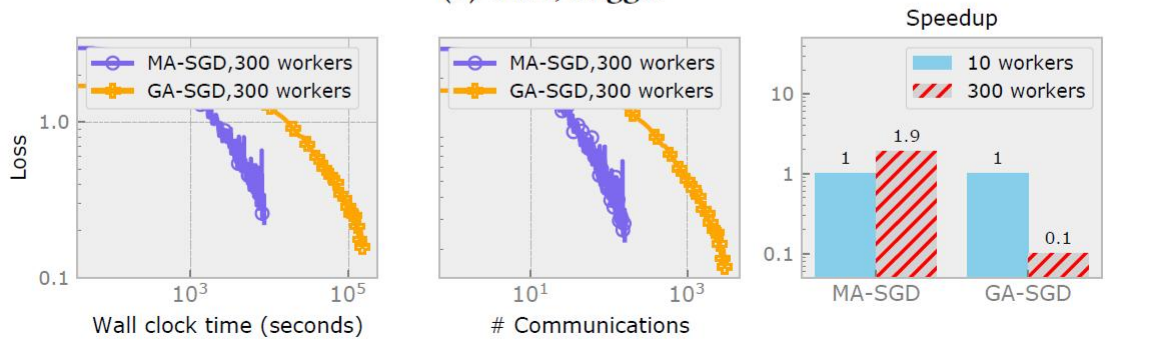
Evaluation: Distributed Optimization Algorithm



(a) LR, Higgs.



(b) SVM, Higgs.



(c) MobileNet, Cifar10.

Evaluation: Communication Channel

- Comparison of S3, Memcached, DynamoDB, and VM-based parameter server.
- A relative cost larger than 1 means S3 is cheaper, whereas a slowdown larger than 1 means S3 is faster.
- DynamoDB cannot handle a large model such as MobileNet.

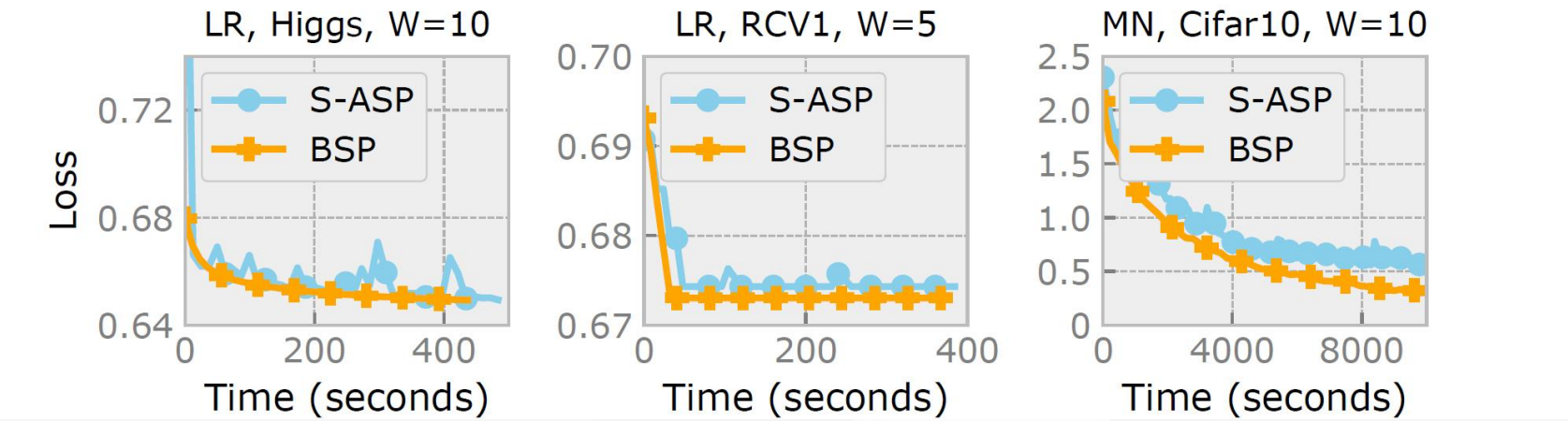
Workload	Memcached vs. S3		DynamoDB vs. S3		VM-PS vs. S3	
	cost	slowdown	cost	slowdown	cost	slowdown
LR,Higgs,W=10	5	4.17	0.95	0.83	4.7	3.85
LR,Higgs,W=50	4.5	3.70	0.92	0.81	4.47	3.70
KMeans,Higgs,W=50,k=10	1.58	1.32	1.13	0.93	1.48	1.23
KMeans,Higgs,W=50,k=1K	1.43	1.19	1.03	0.90	1.52	1.27
MobileNet,Cifar10,W=10	0.9	0.77	N/A	N/A	4.8	4.01
MobileNet,Cifar10,W=50	0.89	0.75	N/A	N/A	4.85	4.03

Evaluation

Communication Patterns

Model & Dataset	Model Size	AllReduce	ScatterReduce
LR,Higgs,W=50	224B	9.2s	9.8s
MobileNet,Cifar10,W=10	12MB	3.3s	3.1s
ResNet,Cifar10,W=10	89MB	17.3s	8.5s

Synchronization Protocols



04

INFless

- INFless: A Native Serverless System for Low-Latency, High-Throughput Inference

INFless' s Overview

- Background: Existing serverless platforms do not cater to the needs of ML inference.
  - do not address the challenge of providing solutions for guaranteeing latency
  - the resource efficiency at the serverless provider side is also very low
- Design Goal: A native serverless inference system introduces several challenges that need to be addressed.
  - Low latency
  - High throughput
  - Low overhead
- Contribution
  - We co-design the **batch** management and heterogeneous resource allocation mechanism, and propose the **non-uniform** scaling policy to maximize resource efficiency.
  - We propose a lightweight **combined operator profiling** method.
  - We design a novel **Long-Short Term Histogram (LSTH)** policy.

Background: Limitations of Existing Serverless Platforms

- Observation #1: High latency
  - The commercial serverless platform lacks the support of accelerators and therefore cannot provide low latency services for large-sized inference models.
- Observation #2: For batch-enabled inference, commercial serverless platforms cannot provide low-latency services for some small-sized models.
- Observation #3: Resource over-provisioning
  - The proportional CPU-memory allocation policy set by a commercial serverless platform does not fit with computationally-intensive inference.
- Observation #4: The “one-to-one mapping” request processing policy of commercial serverless platforms causes low resource utilization.
- Observation #5: OTP batching lacks the codesign of batch configuration, instance scheduling and resource allocation, bringing only limited throughput improvement.

Design: System Architecture

① Function deployment

② DAG structure parsing

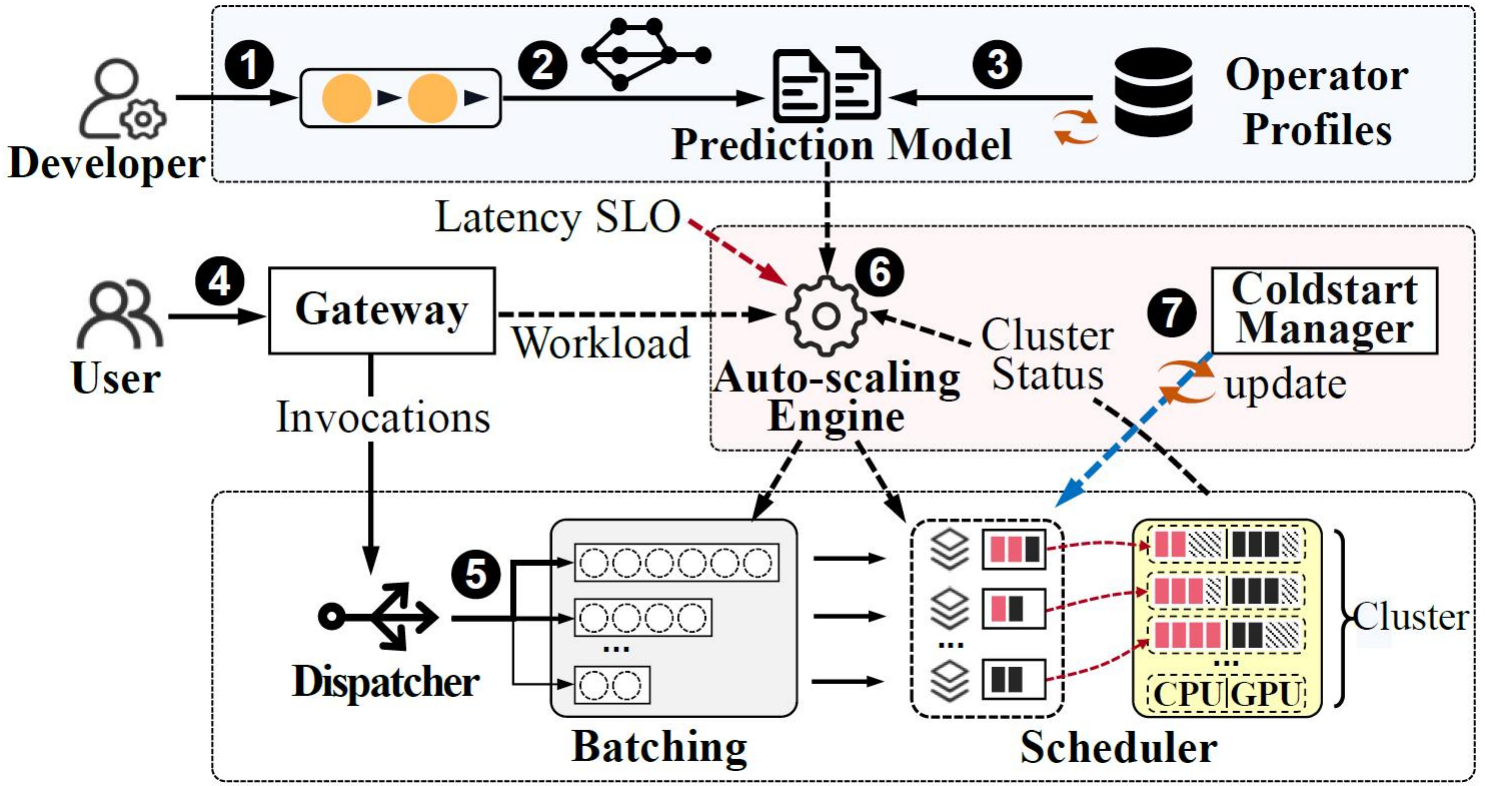
③ Operator profiling

④ Inference query

⑤ Dispatching and batching

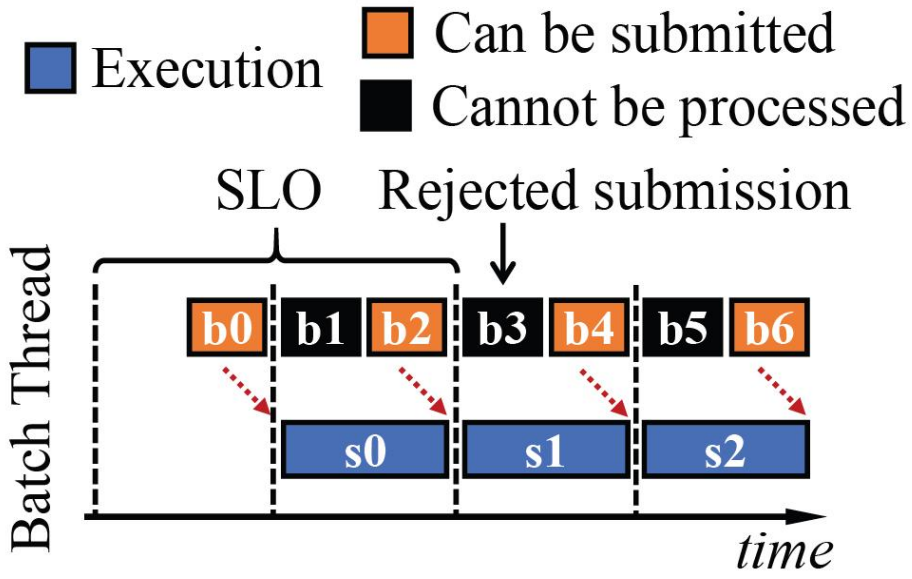
⑥ Resource configuration

⑦ Cold-start avoidance

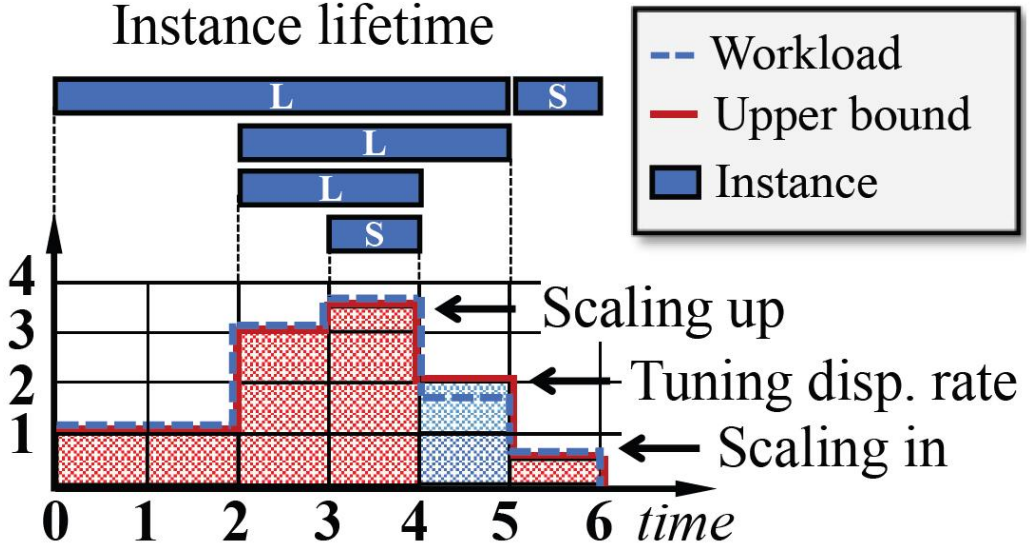


Design: Built-in, Non-Uniform Batching

- Built-in: Batching is integrated into the serverless platform, enabling simultaneous, collaborative control over batch size, resource allocation and placements.
- Non-uniform: Each instance has an individual batch queue to aggregate inference requests.



(a) Over-submission

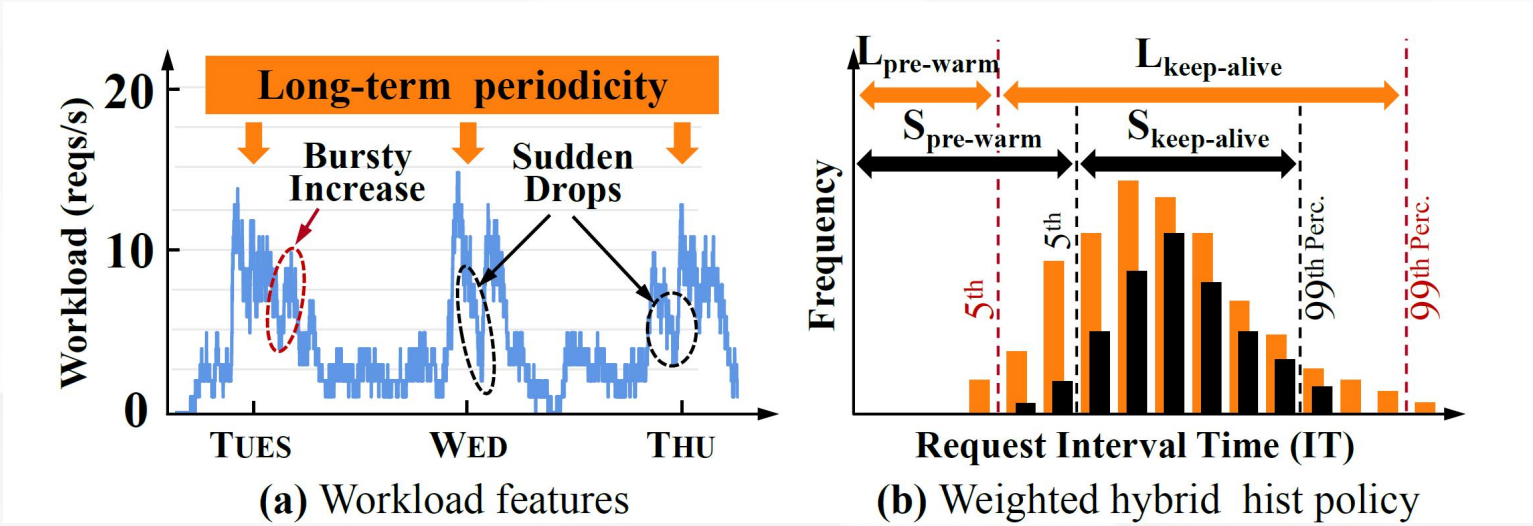


(b) Instance scaling

Design: Managing Cold Starts with LSTH

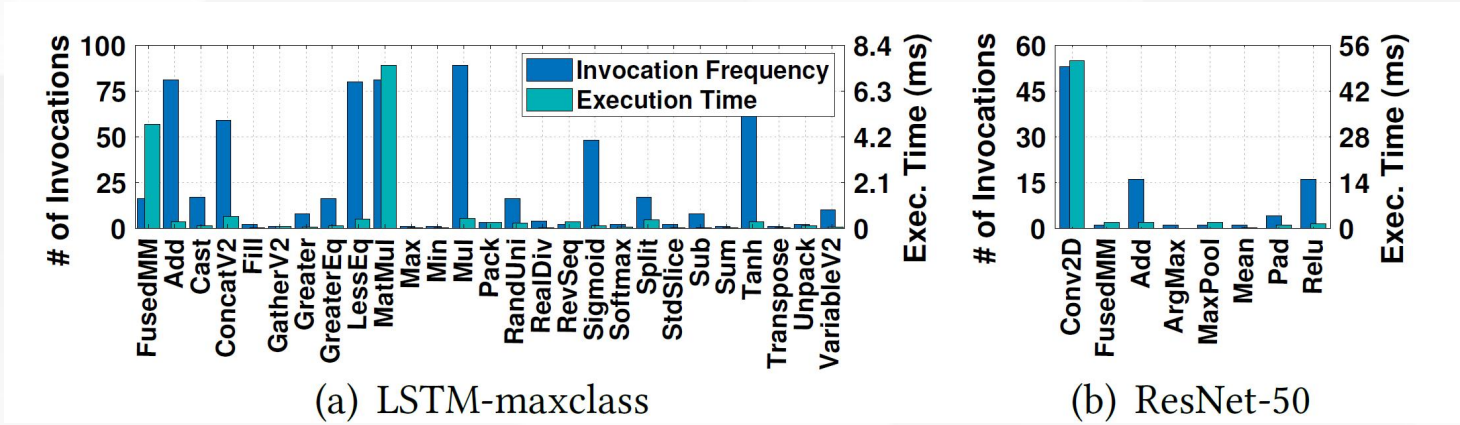
- Long-term periodicity (LTP): the request load shows a diurnal user access pattern overall;
- Short-term burst (STB): there are many sudden changes (including both increases and decreases) in short times.
- Long-Short Term Histogram (LSTH)

$$\text{pre-warm} = \gamma L_{\text{pre-warm}} + (1-\gamma) S_{\text{pre-warm}}$$
$$\text{keep-alive} = \gamma L_{\text{keep-alive}} + (1-\gamma) S_{\text{keep-alive}}$$



Design: Combined Operator Profiling

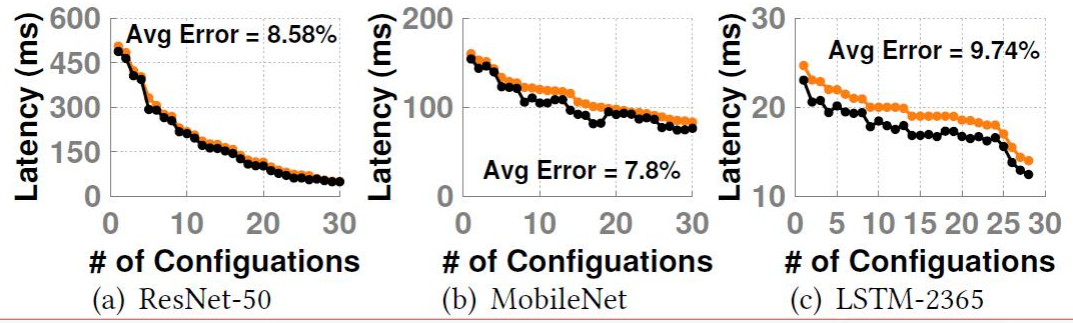
- Observation: Inference functions share a common set of operators, and the execution time is dominated by a small subset of them.



(a) LSTM-maxclass (b) ResNet-50

- Database: build a operator profile database <operator, batch-size, CPU, GPU, time>, and estimate the model execution latency based on the database.

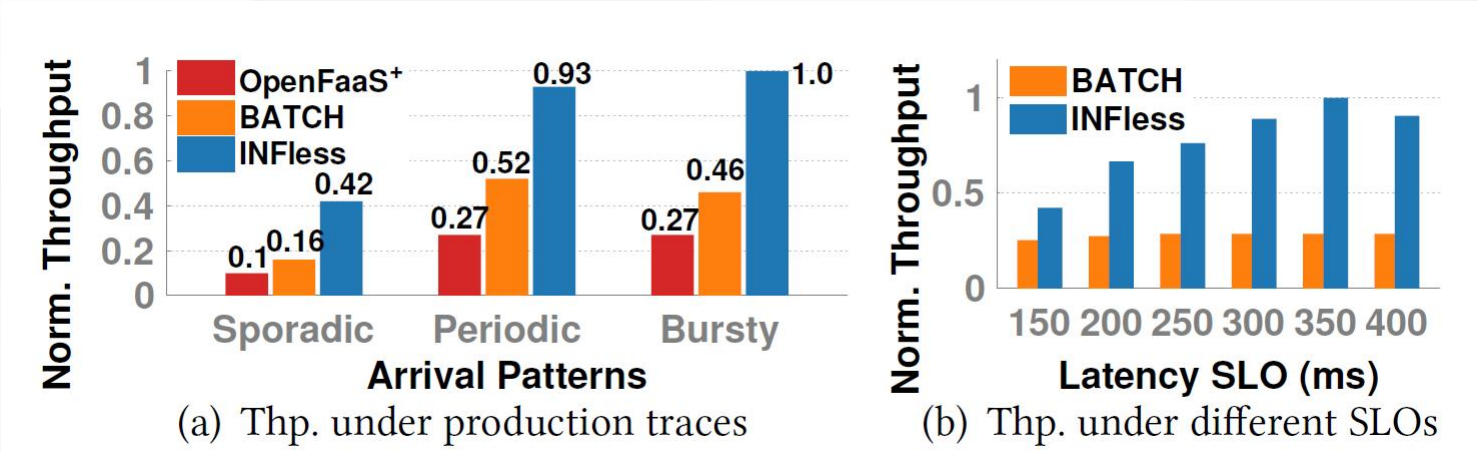
- Result



(a) ResNet-50 (b) MobileNet (c) LSTM-2365

Evaluation: Local Cluster Evaluation

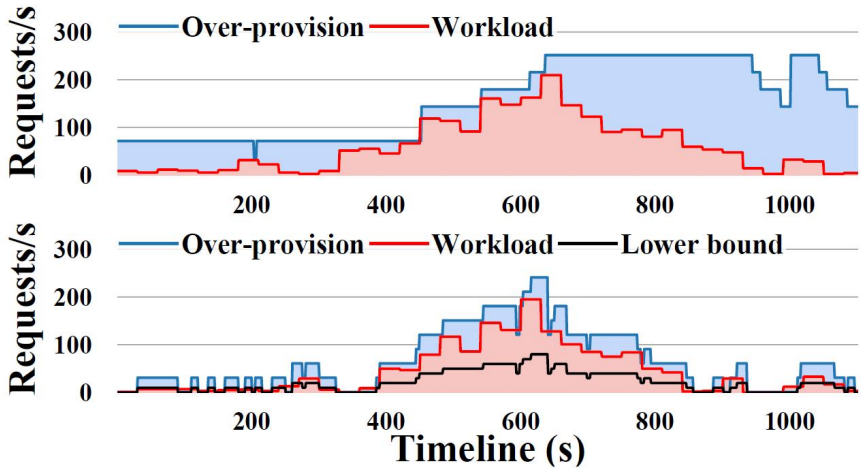
- High throughput: INFless improves system throughput by 2x-5x.



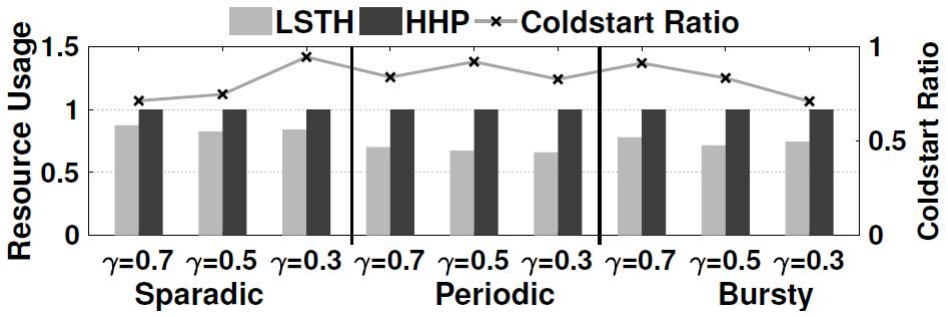
- Component analysis: Every component of INFless contributes much to throughput improvement, with batching being the highest.
- Flexible configurations: INFless opts for flexible configurations on both batch-sizes and resource allocations.

Evaluation: Local Cluster Evaluation

- Less over-provisioning: INFless's resource allocation policy reduces the resource provisioning significantly.

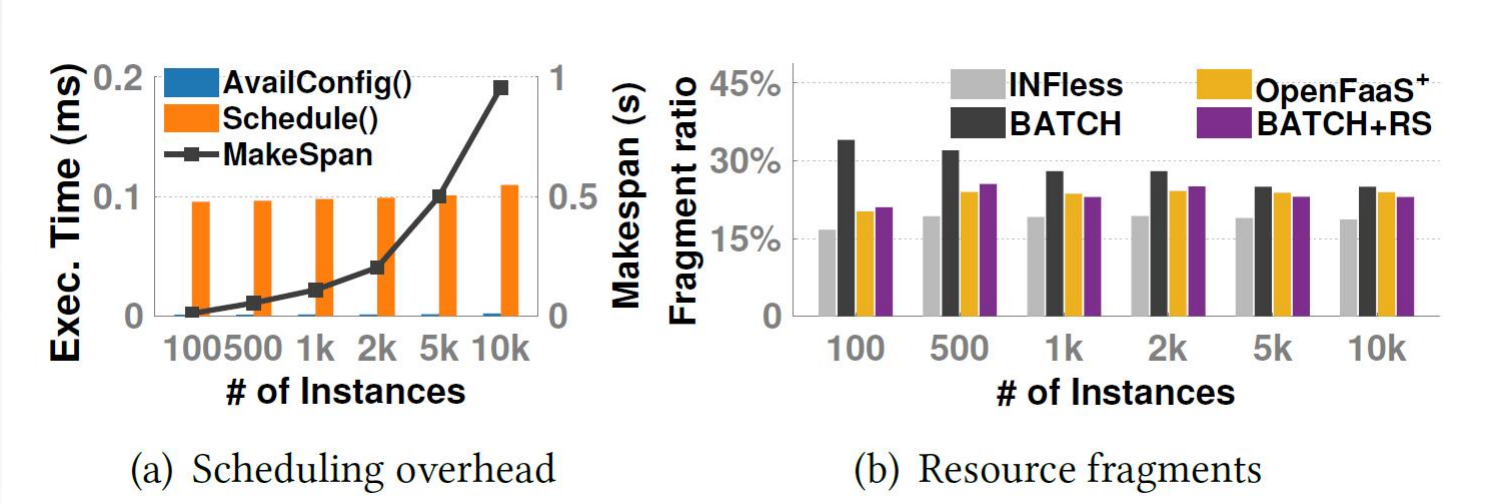


- SLO violation: INFless can guarantee the latency SLO of inference workloads.
- Cold start: Compared with HHP, our LSTH policy can reduce the cold start rate by 20%.



Evaluation: Large Scale Simulation

- Scalability: INFless scales well in large-scale evaluations.
- Resource fragments: INFless's resource-aware scheduling algorithm reduces the resource fragments significantly.



- Cost efficiency: INFless can help service developers and cloud providers reduce the cost of constructing inference services.

05

Conclusion

Serverless and Machine Learning

Paper	Year	Conference	Topic
NumPyWren	2020	SoCC	Matrix computation
Cirrus	2019	SoCC	Model training
LambdaML	2021	SIGMOD	Model training
INFless	2022	ASPLOS	Model inference

谢谢！

饮 水 思 源　爱 国 荣 校