A Practical Introduction to Federated Learning

FederatedScope Team

https://federatedscope.io/

In-Person Presenter: Yaliang Li, Zhen Wang, Bolin Ding

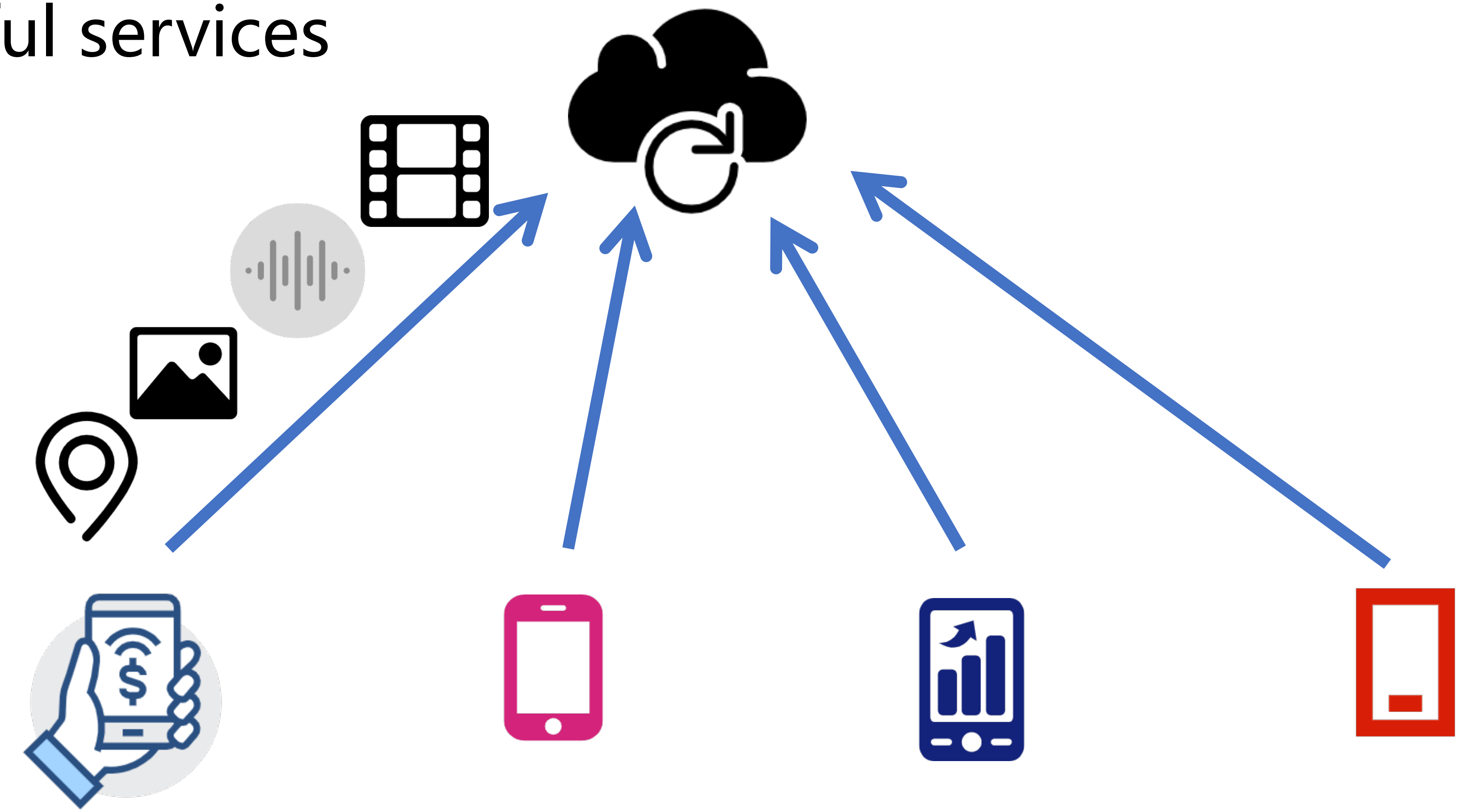
Alibaba Group

Agenda of Tutorial

- Overview
- Personalized Federated Learning
- Federated Graph Learning
- Federated Hyperparameter Optimization
- Privacy Attacks

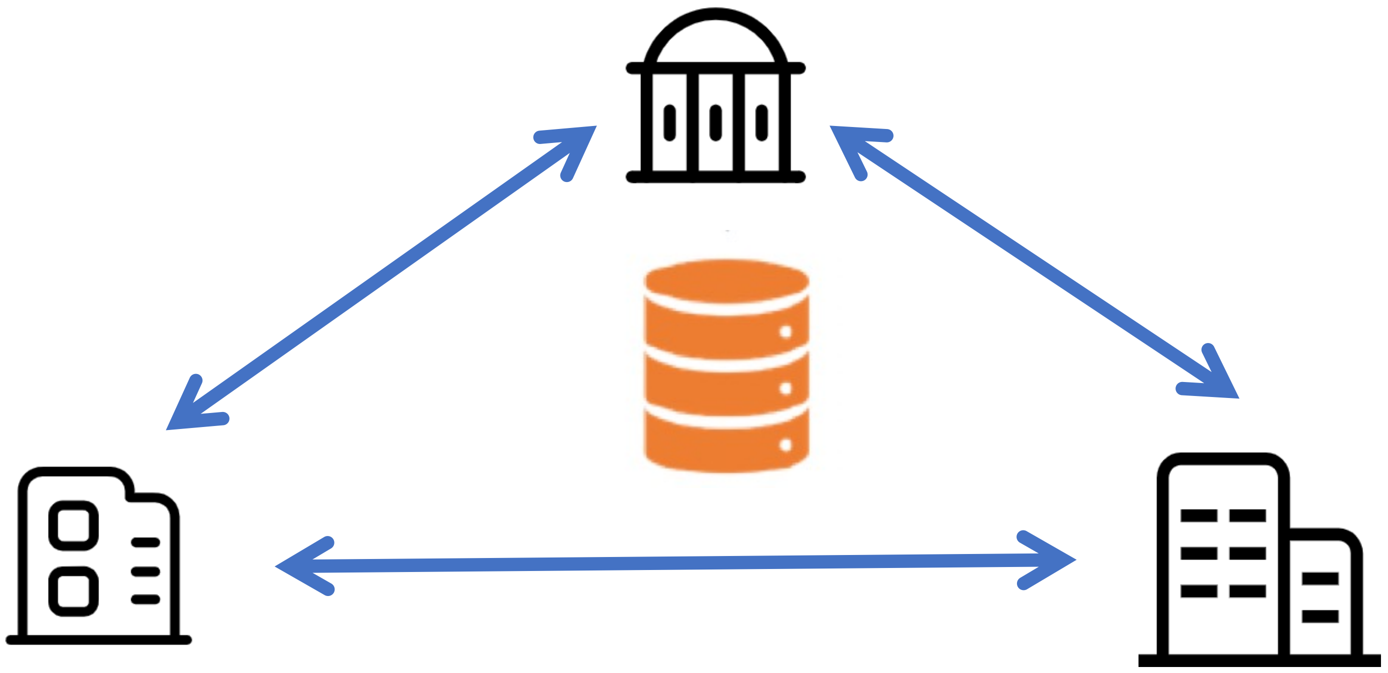
Privacy Leakage in Practice

- App users are required to upload private local data for plentiful services



Privacy Leakage in Practice

- Organizations/companies (such as hospitals and banks) exchange data for research or business purposes



Concerns and Regulations

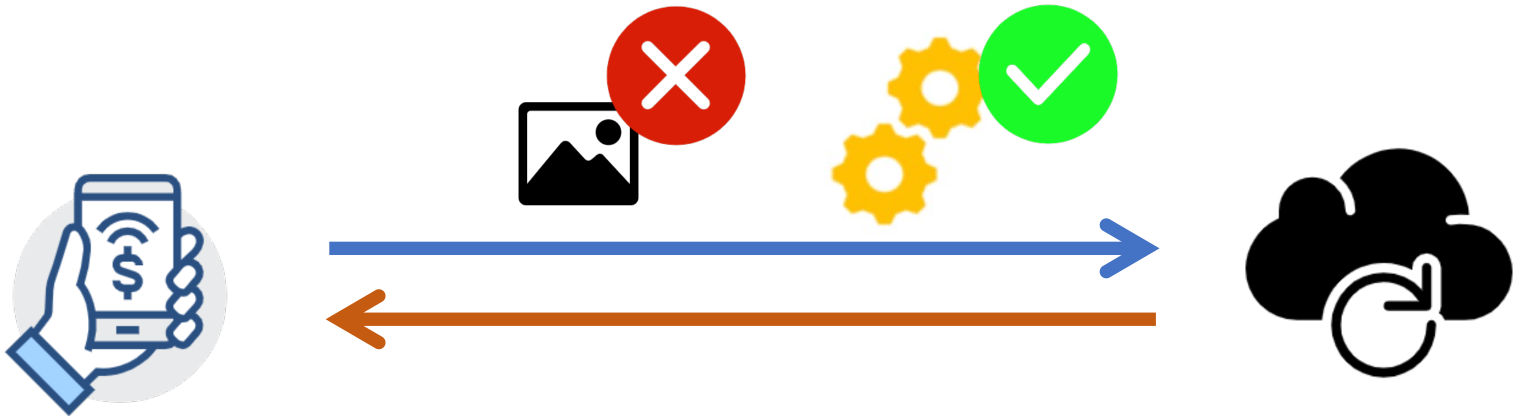
- The public's awareness of privacy protection
- Protection regulations, such as GDPR[1]

Are my private data safe
when being shared out?

[1] General Data Protection Regulation (GDPR). https://gdpr-info.eu

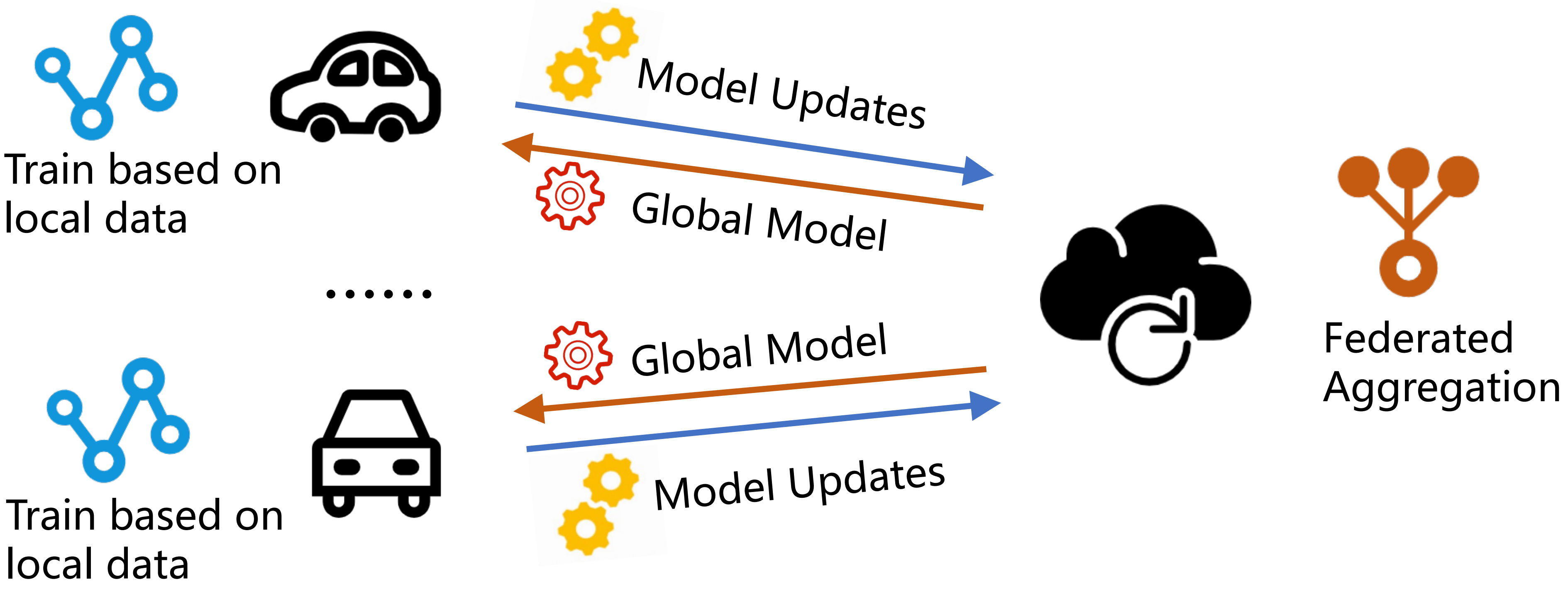
Federated Learning

- Federated Learning is a learning paradigm proposed for collaboratively training models from dispersed data
- Instead of sharing the private data, participants only share the learned knowledge



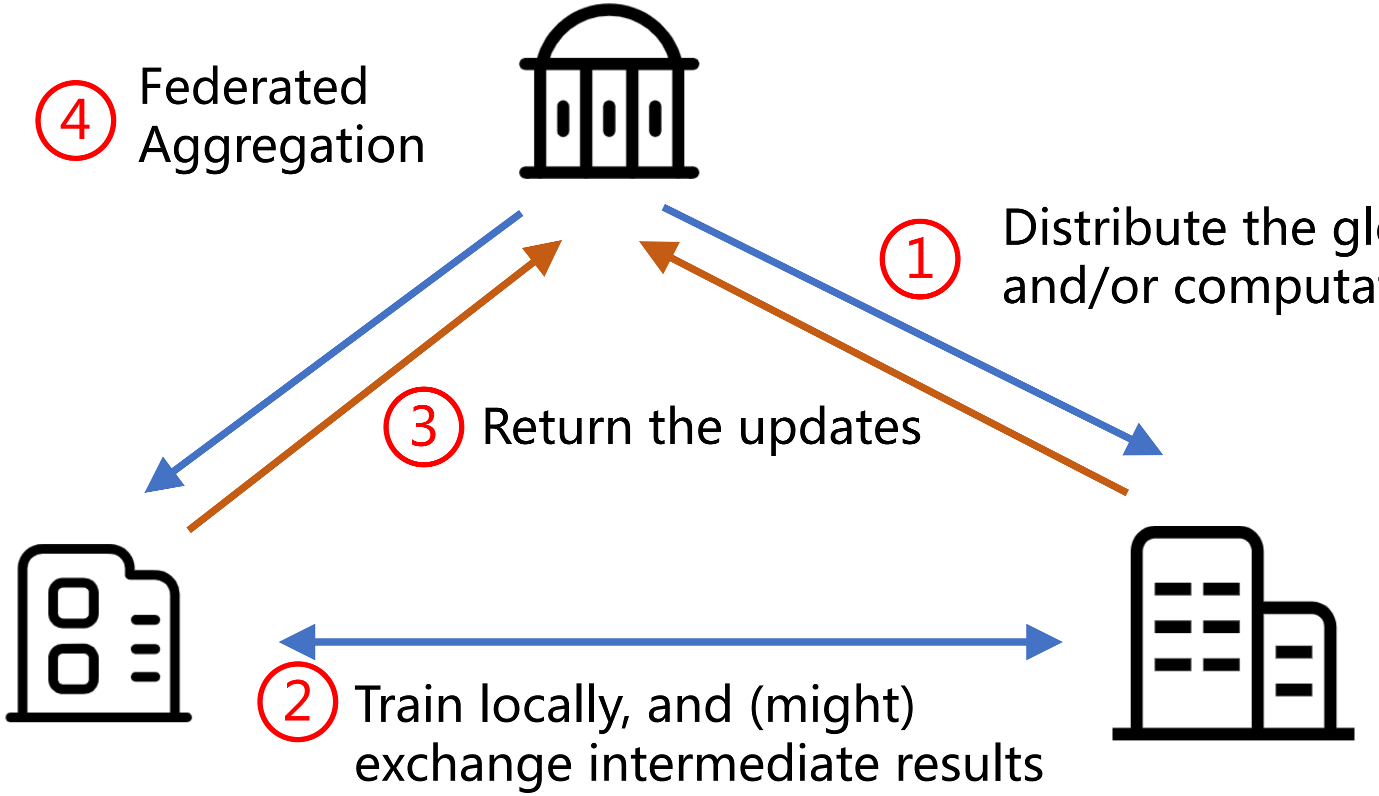
Federated Learning: Cross-device

- A large number of mobile or IoT devices with local stored data collaboratively learn a global model



Federated Learning: Cross-silo

- Multiple parties collaboratively learn global knowledge based on their private data with similar or complementary features



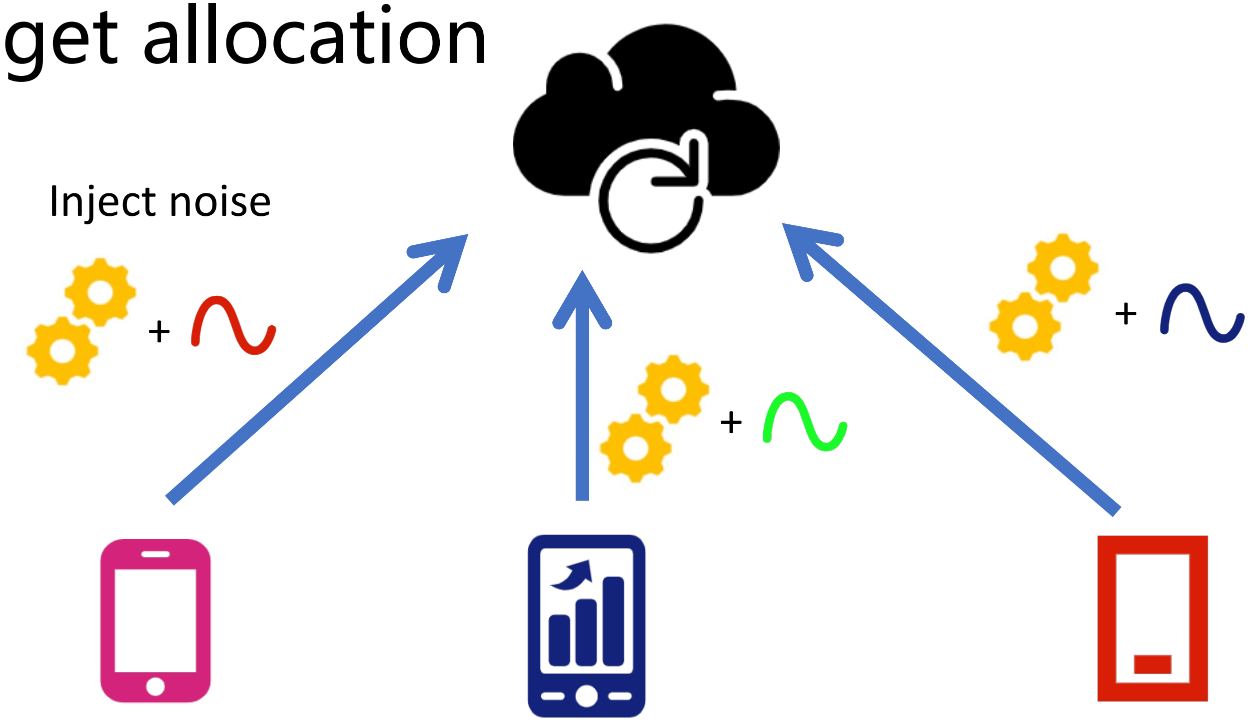
Privacy Protection Techniques

To further satisfy privacy protection requirements, various privacy protection techniques can be integrated into FL:

- Differential Privacy (DP)
- Homomorphic Encryption (HE)
- Secure Multi-Party Computation (MPC)

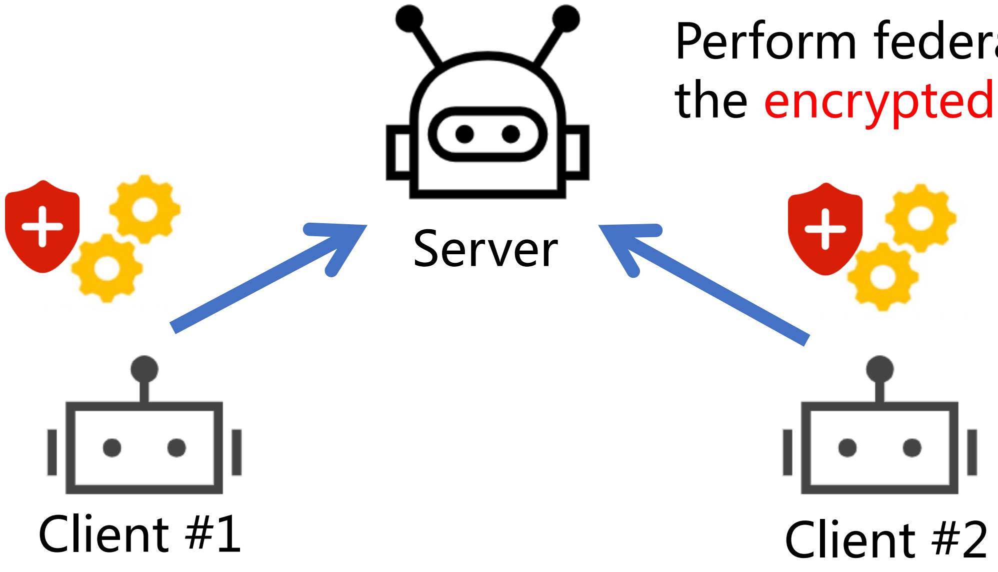
Differential Privacy (DP)

- The information are perturbed before sharing
- Trade-off between privacy protection and model utility
- Privacy budget allocation



Homomorphic Encryption (HE)

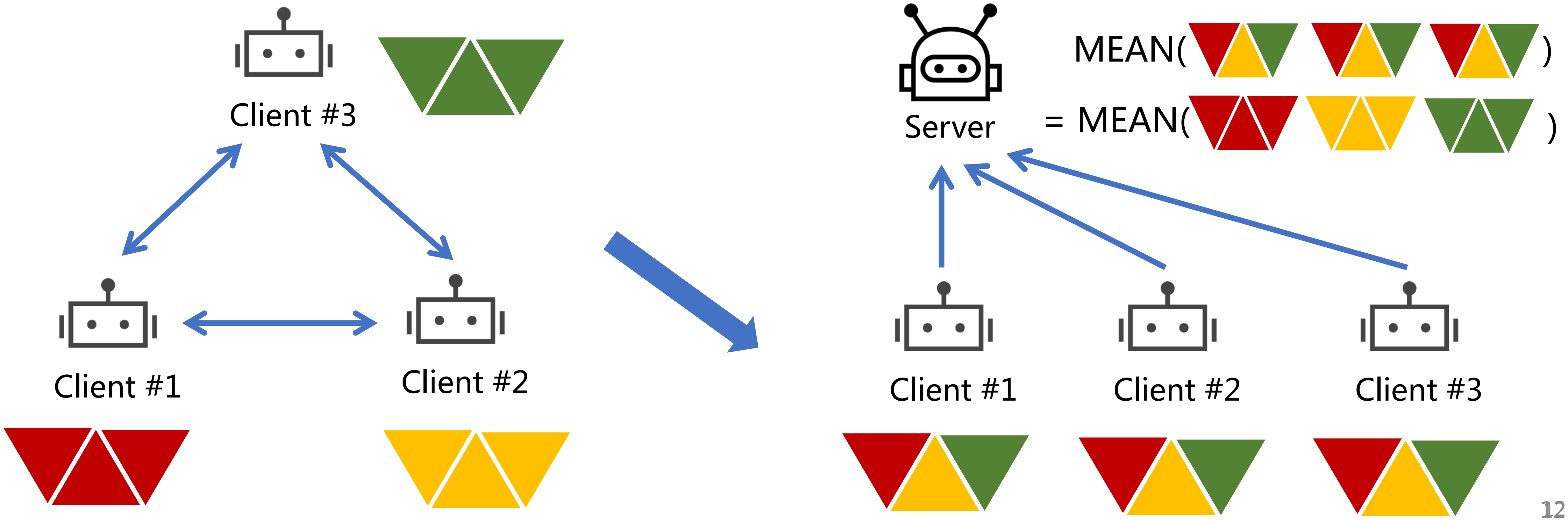
- Participants are allowed to perform computations on encrypted data
- E.g., additively HE: $[a] + [b] = [a + b]$



Perform federated aggregation on the encrypted results

Secure Multi-Party Computation (MPC)

- MPC aims to jointly compute a function by multiple participants while keeping the original inputs private.



Federated Learning Platforms

	TensorFlow Federated	Google
	PySyft	OpenMined
	PaddleFL	Baidu
FATE	FATE	WeBank
FederatedScope	FederatedScope	Alibaba Group
and more … …		

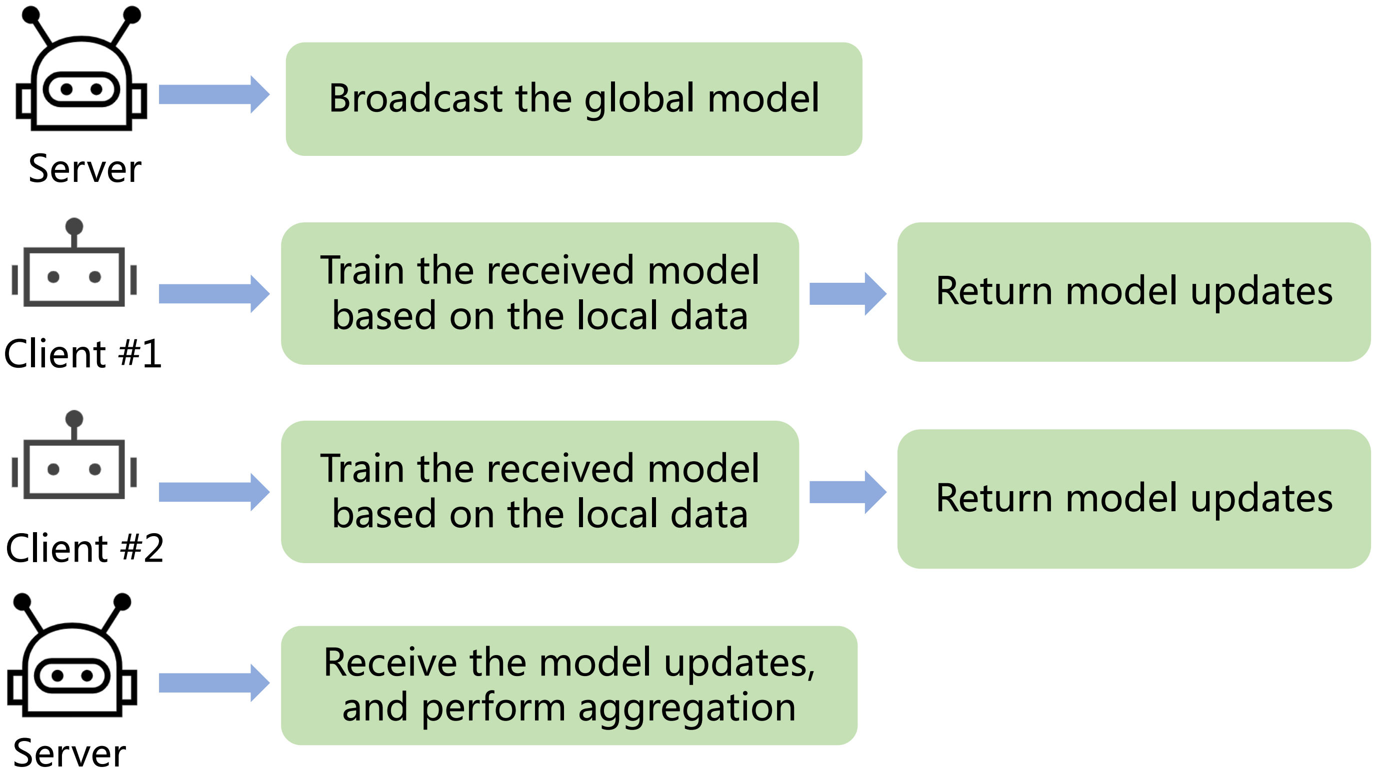
FederatedScope

- FederatedScope[2] is an easy-to-use FL platform which employs an event-driven architecture
- FederatedScope provides users with great flexibility to independently describe the behaviors of different participants, friendly for research

[2] FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. arXiv preprint, 2022.

Event-driven v.s. Procedural

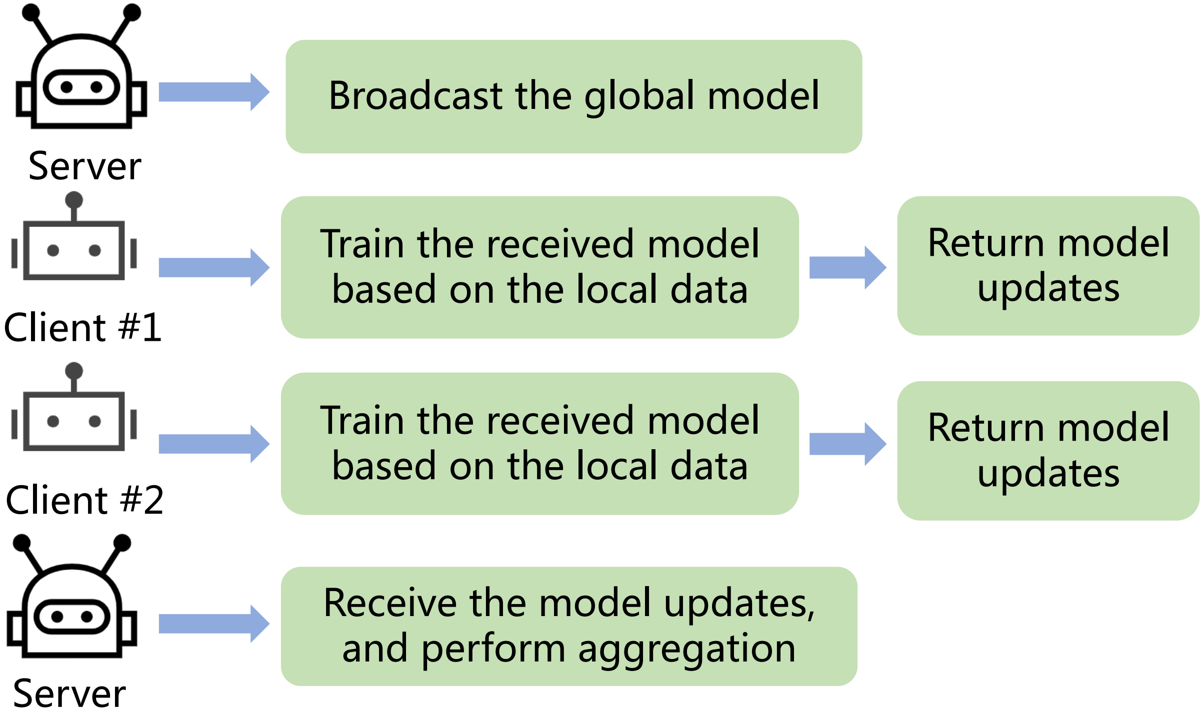
- Take vanilla FedAvg[3] as an example



[3] Communication-efficient learning of deep networks from decentralized data. In AISTATS, 2017.

Event-driven v.s. Procedural

- Take vanilla FedAvg[3] as an example

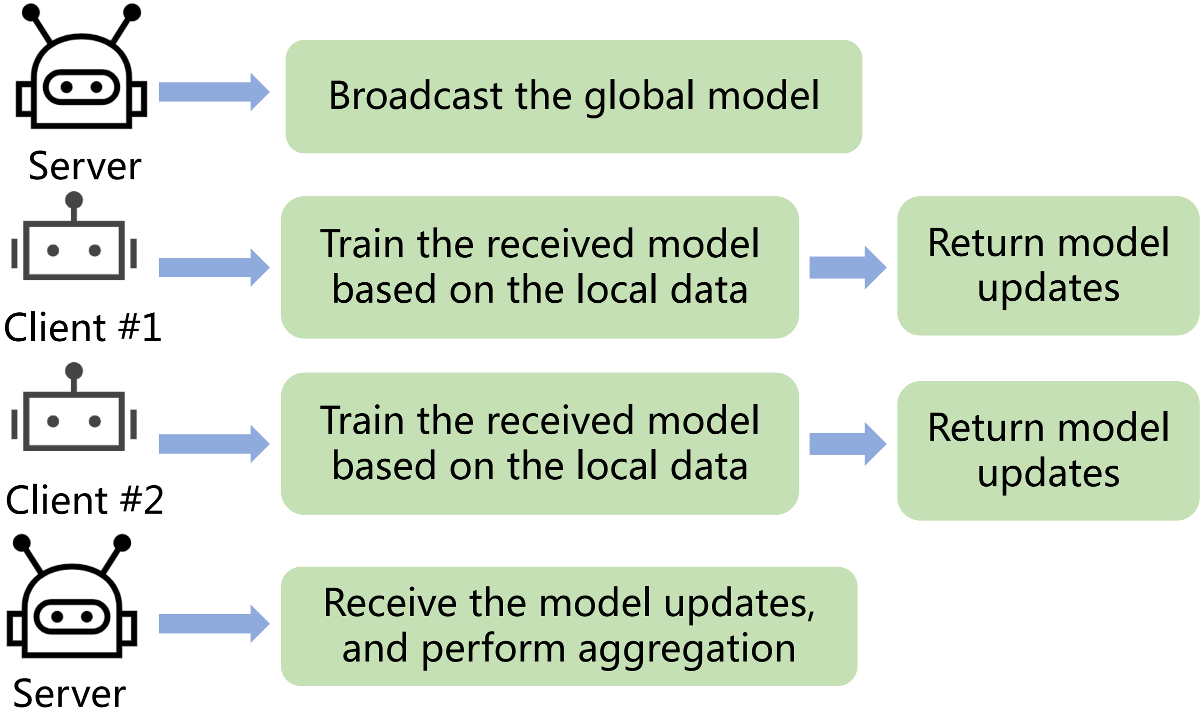


Procedural

Server broadcasts model to Clients;
Client #1 receives the model, trains locally, returns model updates;
Client #2 receives the model, trains locally, returns model updates;
Server receives the model updates, preforms aggregation, and sends global model;

Event-driven v.s. Procedural

- Take vanilla FedAvg[3] as an example



Procedural

Server broadcasts model to Clients;
Client #1 receives the model, trains locally, returns model updates;
Client #2 receives the model, trains locally, returns model updates;
Server receives the model updates, preforms aggregation, and sends global model;

Event-driven

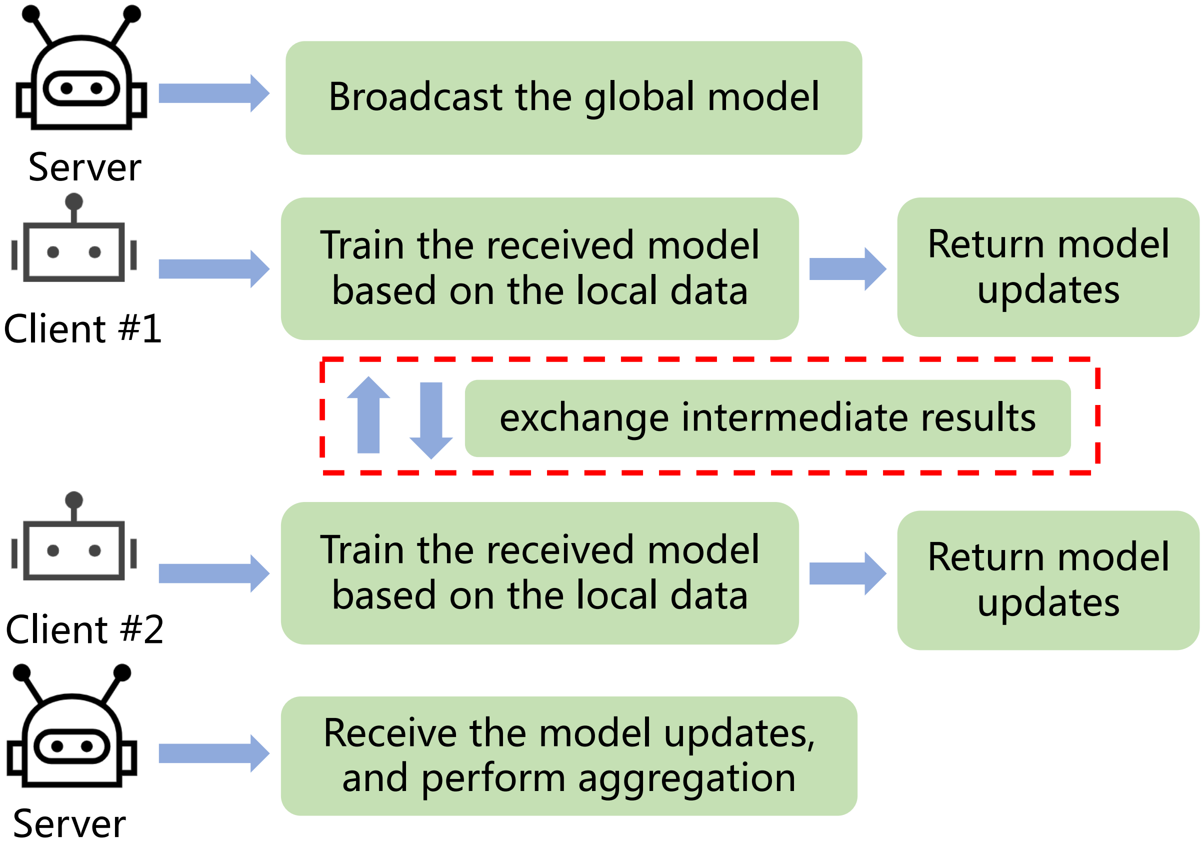
Client:
DEFINE handlers:
- When receiving *model* → Train it on local data and return model updates;

Server:
DEFINE handlers :
- When receiving *model updates* → Preform aggregation and send global model;

Instantiate **Server, Client #1, Client #2; Server** broadcasts the global model.

Event-driven v.s. Procedural

- A simple customization started from FedAvg



Procedural

Server broadcasts model to Clients;
Client #1 receives the model, trains locally~~, returns model updates~~;
Client #2 receives the model, trains locally~~, returns model updates~~;
Client #1 and **Client #2** exchange intermediate results;
Client #1 continues training locally, returns updated model;
Client #2 continues training locally, returns updated model;
Server receives the model updates, preforms aggregation, and sends global model;

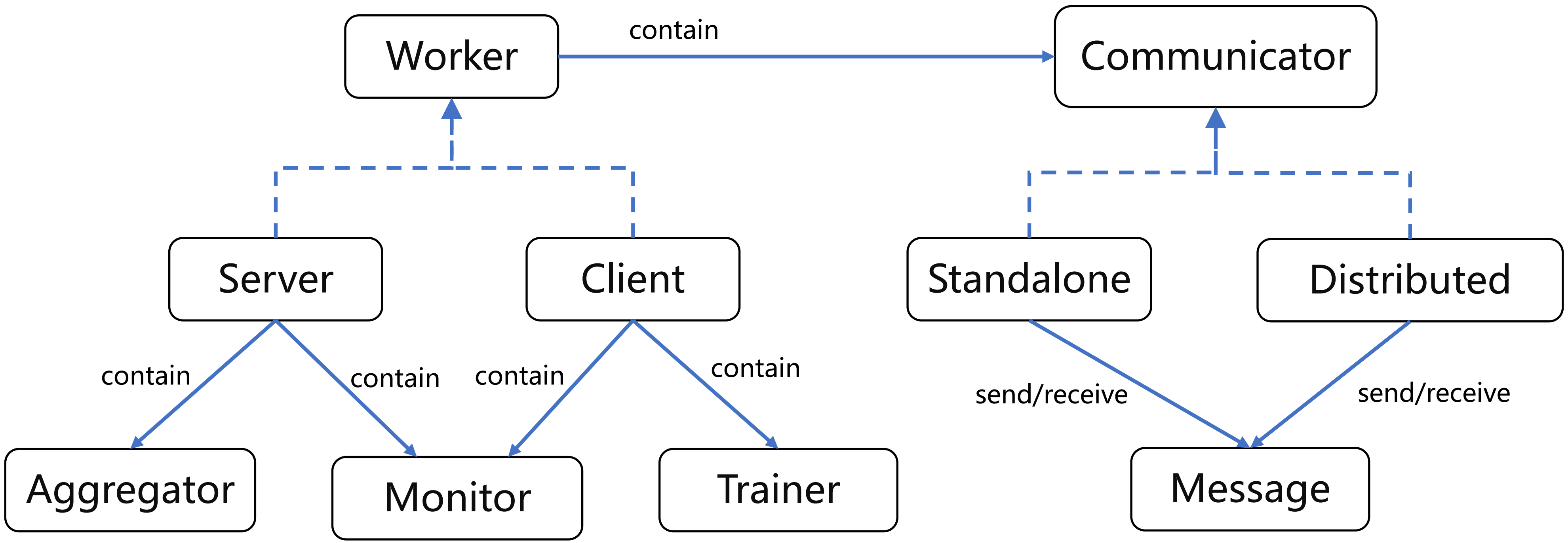
Event-driven

Client:
DEFINE handlers:
- When receiving *model* → Train it on local data, send intermediate results to other clients;
- When receiving *intermediate results* → continues training locally and return model updates;

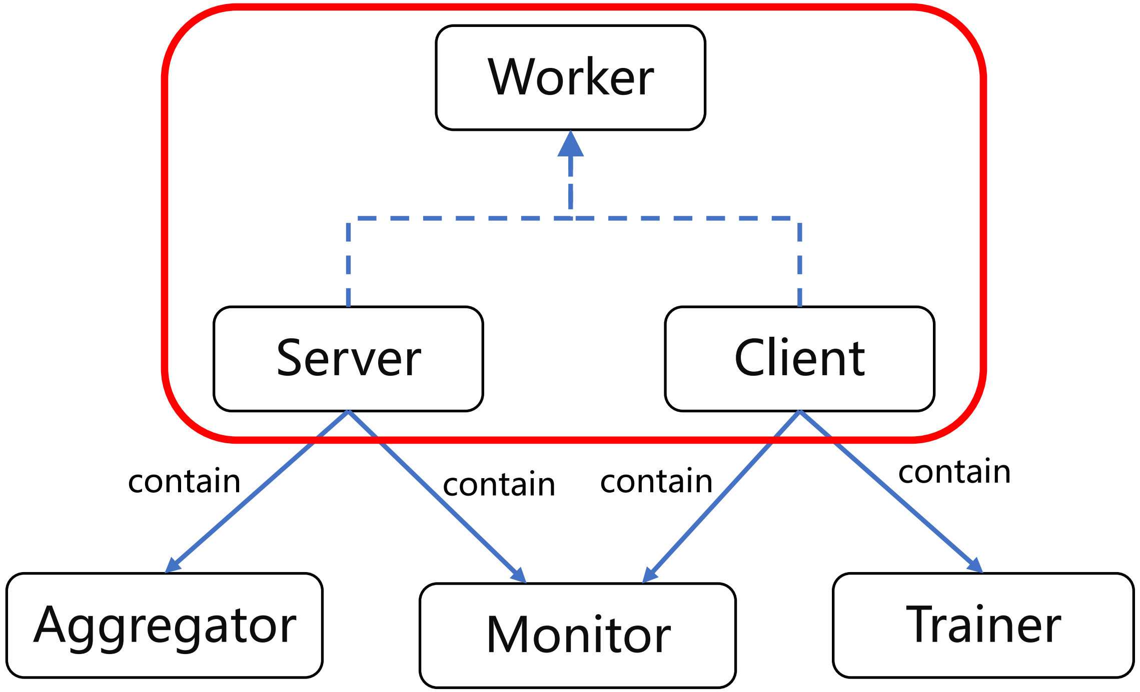
Server:
DEFINE handlers :
- When receiving *model updates* → Preform aggregation and send global model;

Instantiate **Server, Client #1, Client #2; Server** broadcasts the global model.

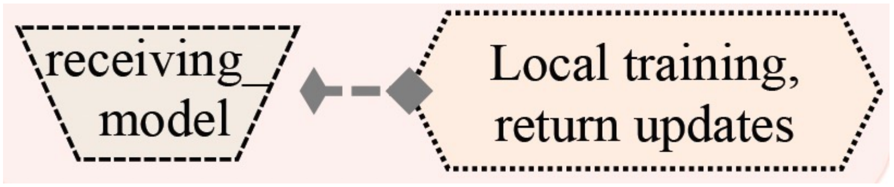
System Design of FederatedScope



Worker Module: Server/Client

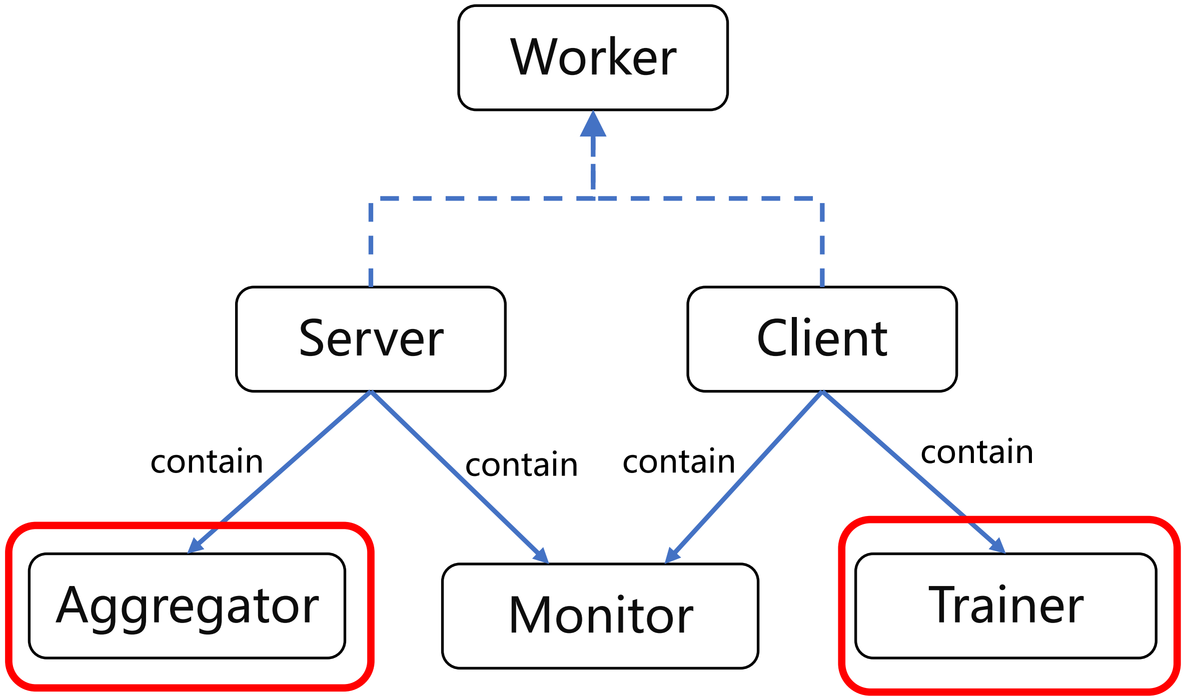


- The behaviors of Server/Client are described via event-handler pairs

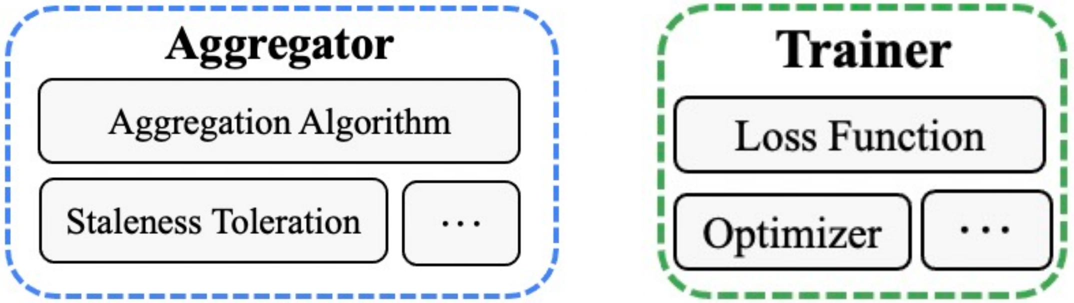


- Server/Client own local data and model
- Server/Client need to exchange messages during the training process

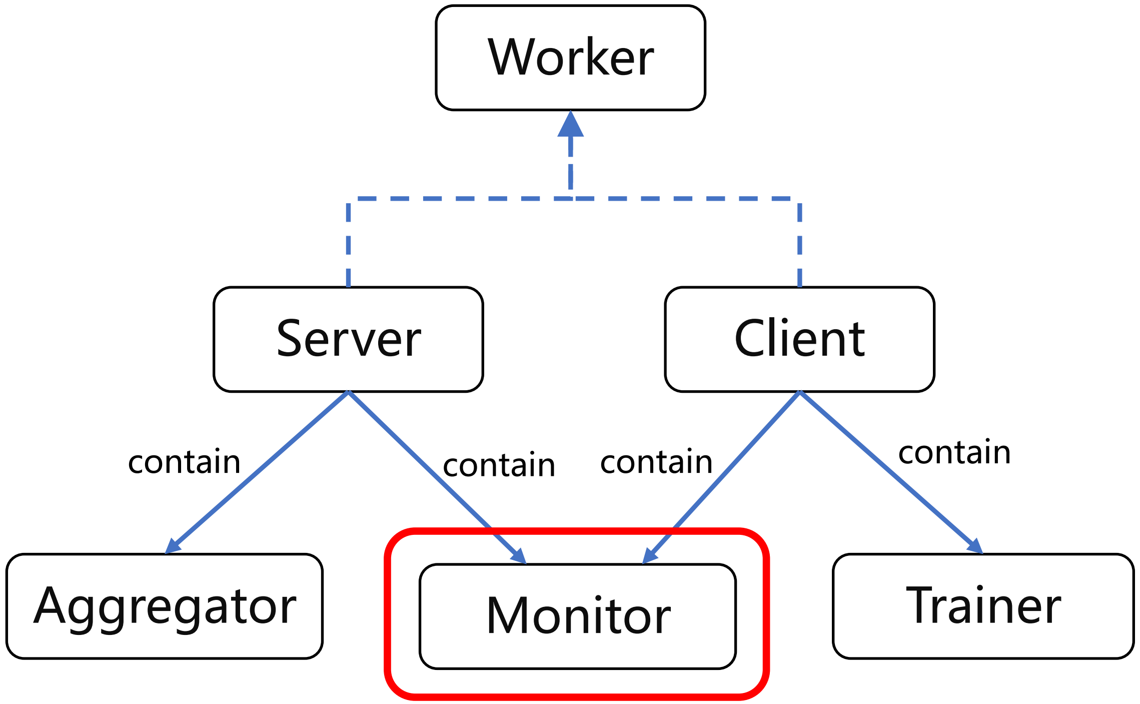
Worker Module: Trainer/Aggregator



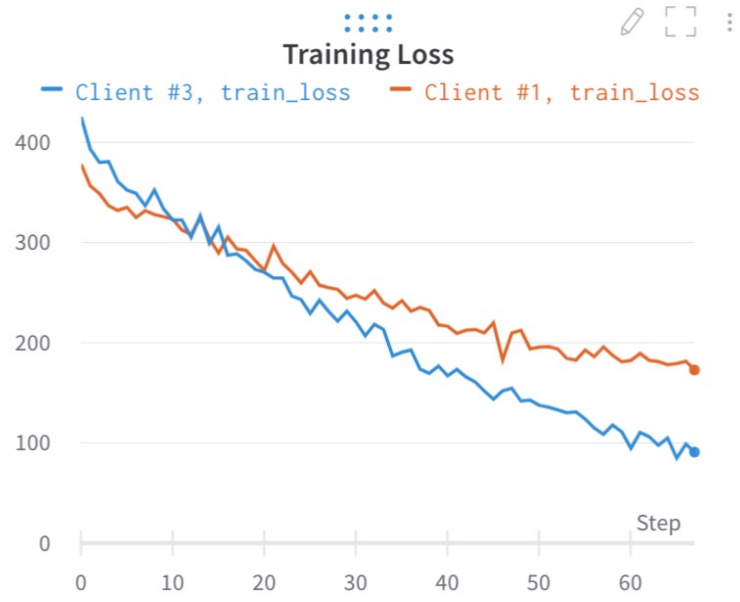
- Client performs local training via Trainer; Server performs aggregation via Aggregator
- Trainer/Aggregator encapsulate the algorithm details, which are entirely decoupled from the federated behaviors of Server/Client



Worker Module: Monitor

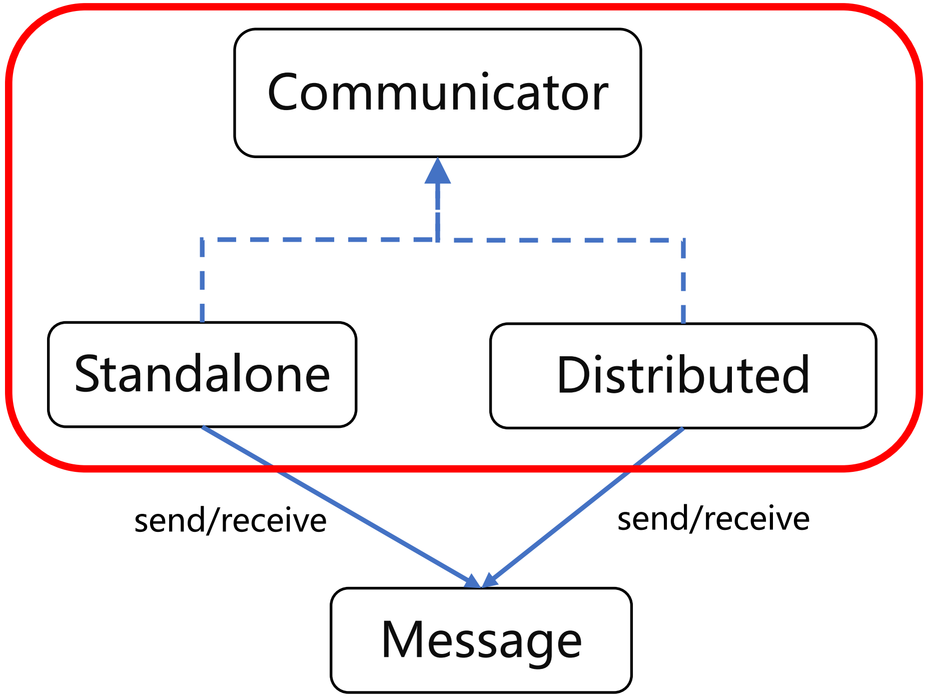


- Monitor are used to record and report the training logs and evaluation metrics
- Both client-wise and global results can be visualized



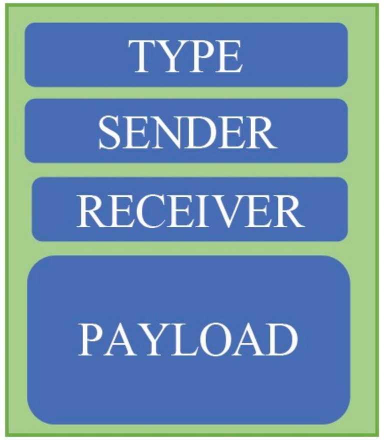
Communication Module: Communicator

- Communicator supports the message exchanging among workers, which encapsulates the communication details
- Communicator provides a unified interface for standalone simulation and distributed deployment

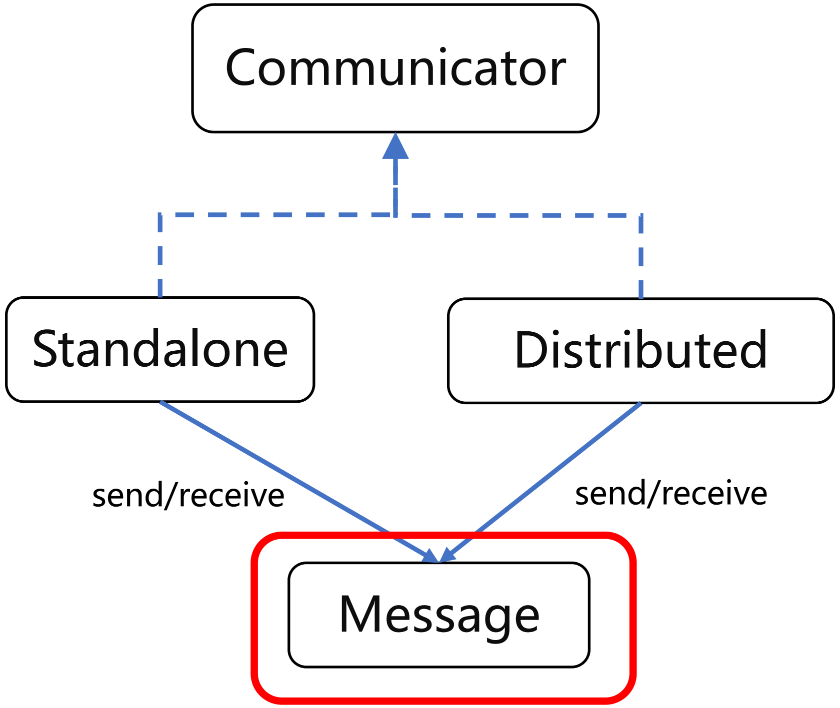


Communication Module: Message

- The exchanged information among workers are abstracted as messages
- Receiving different type of messages (i.e., events) might trigger different handing actions (i.e., handler)



Message



Hands-On Practice with FederatedScope

Developers can implement vanilla FedAvg with FederatedScope as following steps:

- Describe behaviors of clients/server via event-handler pairs
  - When receiving global model -> Perform local training and return updates

```
class Client(object):
    def handler_for_receiving_models(args):
        # Perform local training when receiving the global models
        model_update = trainer.train(model=args.model, data=args.data)
        # Return the model updates to the server
        communicator.send(message=model_update, receiver=server)
```

Hands-On Practice with FederatedScope

Developers can implement vanilla FedAvg with FederatedScope as following steps:

- Describe behaviors of clients/server via event-handler pairs
  - When receiving updates -> Save the updates, check aggregation condition
  - When achieving aggregation goal -> Perform aggregation, and start a training round or terminate the training

```python
class Server(object):
    def handler_for_receiving_updates(args):
        # Save the received updates
        msg_buffer.append(args.message)
        # Perform federated aggregation if the aggregation condition
        # has been satisfied
        if check_aggregation_goal() == True:
            updated_model = aggregator.aggregate(updates=msg_buffer)
            # Start a new training round if not termination
            if check_termination() == False:
                communicator.send(message=updated_model, receiver=clients)
```

Hands-On Practice with FederatedScope

Developers can implement vanilla FedAvg with FederatedScope as following steps:

- Describe behaviors of clients/server via event-hander pairs
- Specify the details of local training and federated aggregation
  - The implementation of trainer is similar to that of centralized training
  - Decoupled with the federated behaviors of clients/server

```python
class Trainer(object):
    ... ...
    # Describe training behaviors (same as centralized training)
    def train(received_models, data):
        # Personalized algorithms might be applied here
        local_model = update_from_global_models(received_models)
        preds = local_model.forward(data.x)
        args = [optimizer, loss_function, regularizer, ...]
        model_updates = local_model.backward(data.y, preds, args)
        return model_updates
```

Hands-On Practice with FederatedScope

Developers can implement vanilla FedAvg with FederatedScope as following steps:

- Describe behaviors of clients/server via event-hander pairs
- Specify the details of local training and federated aggregation
- Construct FL course

```
class Fed_Runner(object):
    ... ...
    def standalone_set_up(config):
        server = set_up_server(config)
        for client in clients:
            client = set_up_client(config)

    def run():
        for client in clients:
            clients.join_in()
```

Hands-On Practice with FederatedScope

Developers can implement vanilla FedAvg with FederatedScope as following steps:

- Describe behaviors of clients/server via event-hander pairs
- Specify the details of local training and federated aggregation
- Construct FL course

FederatedScope has provided rich implementation of existing FL algorithms, which allows users to conveniently apply them via easy configuring.

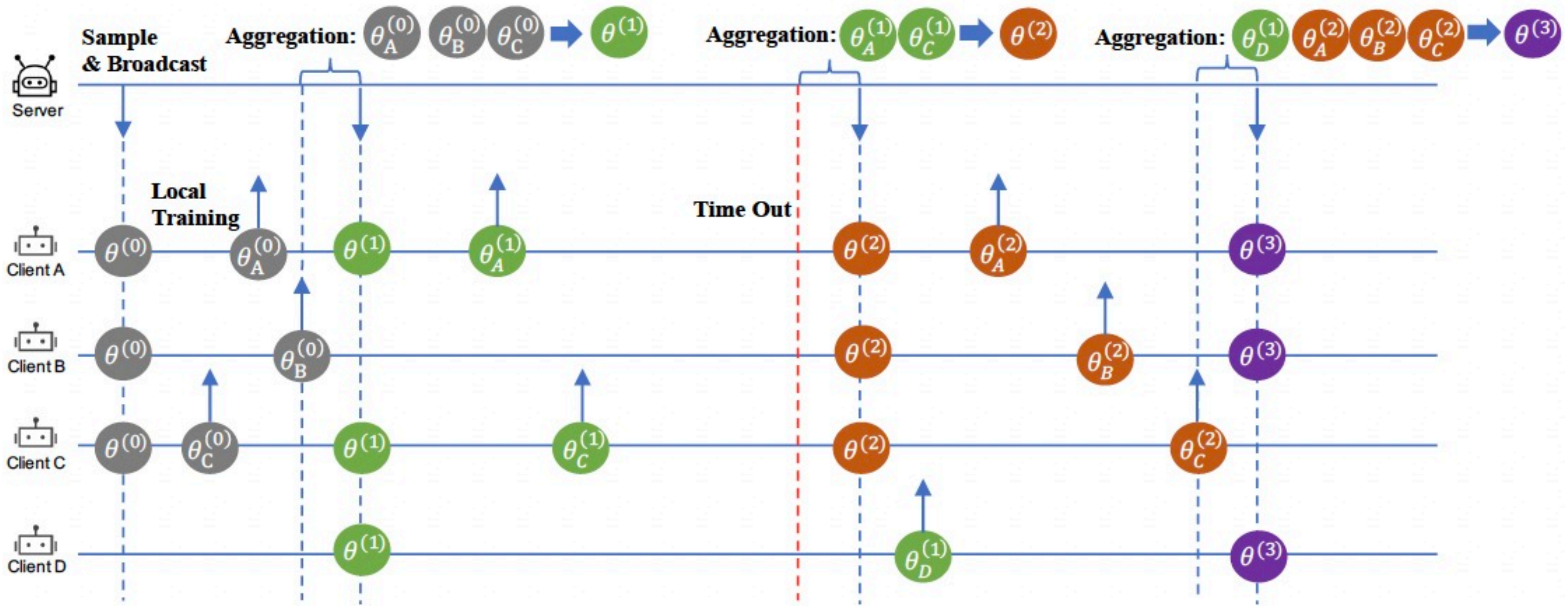
```
cfg.use_gpu = True
cfg.seed = 2
cfg.federate.mode = 'standalone'
cfg.federate.method = 'FedAvg'
cfg.federate.client_num = 5
cfg.federate.total_round_num = 100

cfg.data.type = 'femnist'
cfg.model.type = 'convnet2'

cfg.trainer.type = 'cvtrainer'
cfg.train.optimizer.lr = 0.001
cfg.train.optimizer.weight_decay = 0.0
cfg.grad.grad_clip = 5.0
cfg.criterion.type = 'CrossEntropyLoss'
```

More Advantages of FederatedScope

- Asynchronous FL
- Personalization & Multiple Goals
- Cross-backend FL
- Privacy Protection Techniques

Asynchronous FL

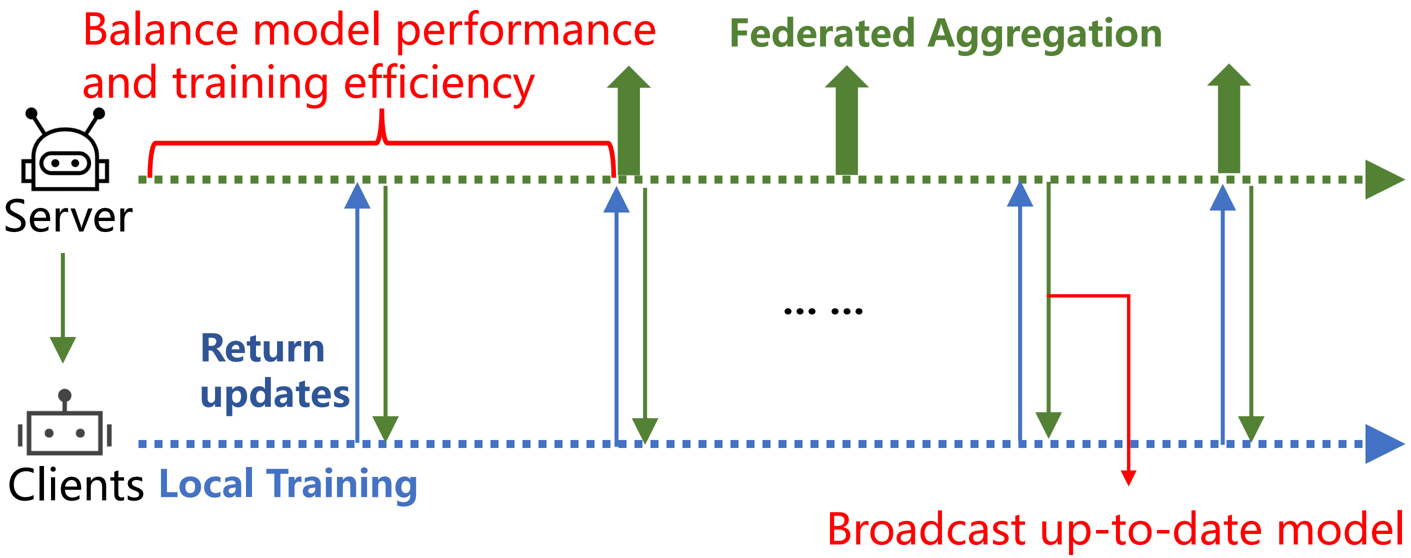
Asynchronous training strategies are important to balance the model performance and training efficiency



Asynchronous FL

The unique behaviors of participants in asynchronous FL are modularized and provided in FederatedScope

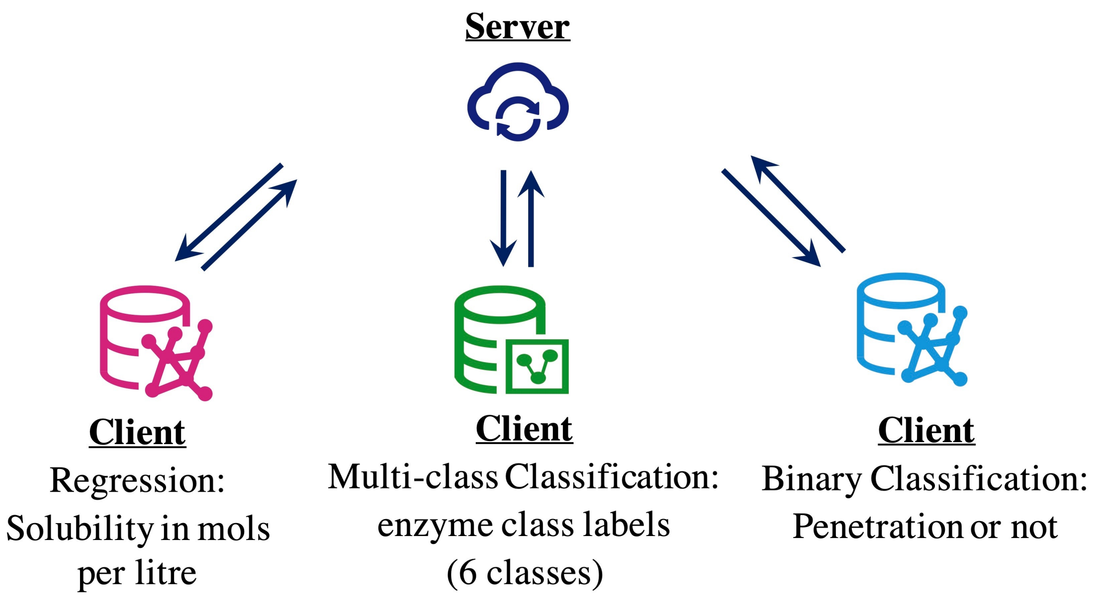
- Staleness toleration
- Broadcasting manner
- Client sampling



Personalization & Multiple Goals

FederatedScope gives participants the right to describe their behaviors from their respective perspectives:

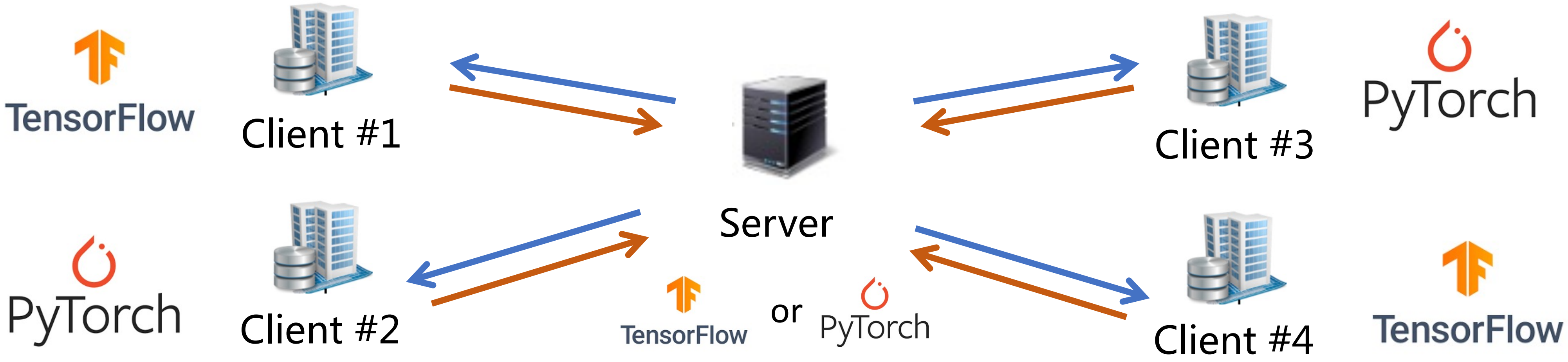
- Client-specific training configurations
- Diverse local training process
- Different learning goals



Cross-backend FL

FederatedScope supports cross-backend FL via message translation:

- Before sharing the messages, participants transform the messages into the pre-defined backend-independent format
- Once the messages are received, the participants parse the messages according their running backends



Privacy Protection Techniques

- For applying DP in FL, FederatedScope provides:
  - plugin operations, such as gradient clipping and noise injecting
  - implementation of state-of-the-art algorithms, such as NbAFL[4]

```
class Client(object):
    def handler_for_receiving_models(args):
        ... ...
        if config.inject_noise_before_sharing == True:
            # Inject certain noise before sharing the message
            args = [noise_distribution, budget, ...]
            protected_messages = add_noise(messages, args)
            send(message=protected_messages, receiver=server)
        else:
            send(message=messages, receiver=server)
```

Privacy Protection Techniques

- For applying DP in FL, FederatedScope provides:
  - plugin operations, such as gradient clipping and noise injecting
  - Implementation of state-of-the-art algorithms, such as NbAFL[4]
- Users can combine different behaviors together to implement more fancy DP algorithms

[4] Federated Learning With Differential Privacy: Algorithms and Performance Analysis. In IEEE TIFS, 2020.

.

Privacy Protection Techniques

- Apply Secret Sharing in FedAvg

```python
class Client(object):
    ... ...
    def handler_for_receiving_models(args):
        # When receiving the global model from the server
        model_updates = trainer.train(model=model, data=data)
        if config.use_ss == True:
            # Broadcast the secret fragments
            secret_frames = secret_split(model_updates)
            broadcast(messages=secret_frames[1:], receiver=clients)
        ... ...

    def handler_for_receiving_secret_fragments(args):
        # When receiving the secret fragments from other clients
        msg_buffer.append(args.message)
        if check_all_frag_received():
            # Mix up the received secret fragments
            mixed_secret = mixup_secret(msg_buffer)
            send(message=mixed_secret, receiver=server)
```

Split the shared message into fragments and broadcast to other clients;

Mix the received frames before sending them to the server;

Implementation examples

Refer to [FederatedScope Playground]() for more examples

1. Prepare datasets: Developers can conveniently conduct experiments on the provided dataset

```
cfg.data.type = 'femnist'
cfg.data.splits = [0.6, 0.2, 0.2]
cfg.data.batch_size = 10
cfg.data.subsample = 0.05
cfg.data.transform = [['ToTensor'], ['Normalize', {'mean': [0.1307], 'std': [0.3081]}]]
```

Implementation examples

Refer to FederatedScope Playground for more examples

2. Prepare models: Developers can set up cfg.model.type = MODEL_NAME to apply a specific model architecture in FL tasks

```
cfg.model.type = 'convnet2'
cfg.model.out_channels = 62
```

Implementation examples

Refer to [FederatedScope Playground]() for more examples

3. Task-specific configuration

```
cfg.use_gpu = False
cfg.eval.freq = 10
cfg.eval.metrics = ['acc', 'loss_regular']

cfg.federate.mode = 'standalone'
cfg.federate.local_update_steps = 5
cfg.federate.total_round_num = 20
cfg.federate.sample_client_num = 5
cfg.federate.client_num = 10

cfg.train.optimizer.lr = 0.001
cfg.train.optimizer.weight_decay = 0.0
cfg.grad.grad_clip = 5.0

cfg.criterion.type = 'CrossEntropyLoss'
cfg.trainer.type = 'cvtrainer'
cfg.seed = 123
```

Implementation examples

Refer to [FederatedScope Playground]() for more examples

4. Enjoy your journey of Federated Learning!

```
from federatedscope.core.fed_runner import FedRunner
from federatedscope.core.auxiliaries.worker_builder import get_server_cls, get_client_cls

Fed_runner = FedRunner(data=data,
                       server_class=get_server_cls(cfg),
                       client_class=get_client_cls(cfg),
                       config=cfg.clone())
Fed_runner.run()
```

References

[1] General Data Protection Regulation (GDPR). https://gdpr-info.eu

[2] FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. arXiv preprint, 2022.

[3] Communication-efficient learning of deep networks from decentralized data. In AISTATS, 2017.

[4] Federated Learning With Differential Privacy: Algorithms and Performance Analysis. In IEEE TIFS, 2020.

Agenda of Tutorial

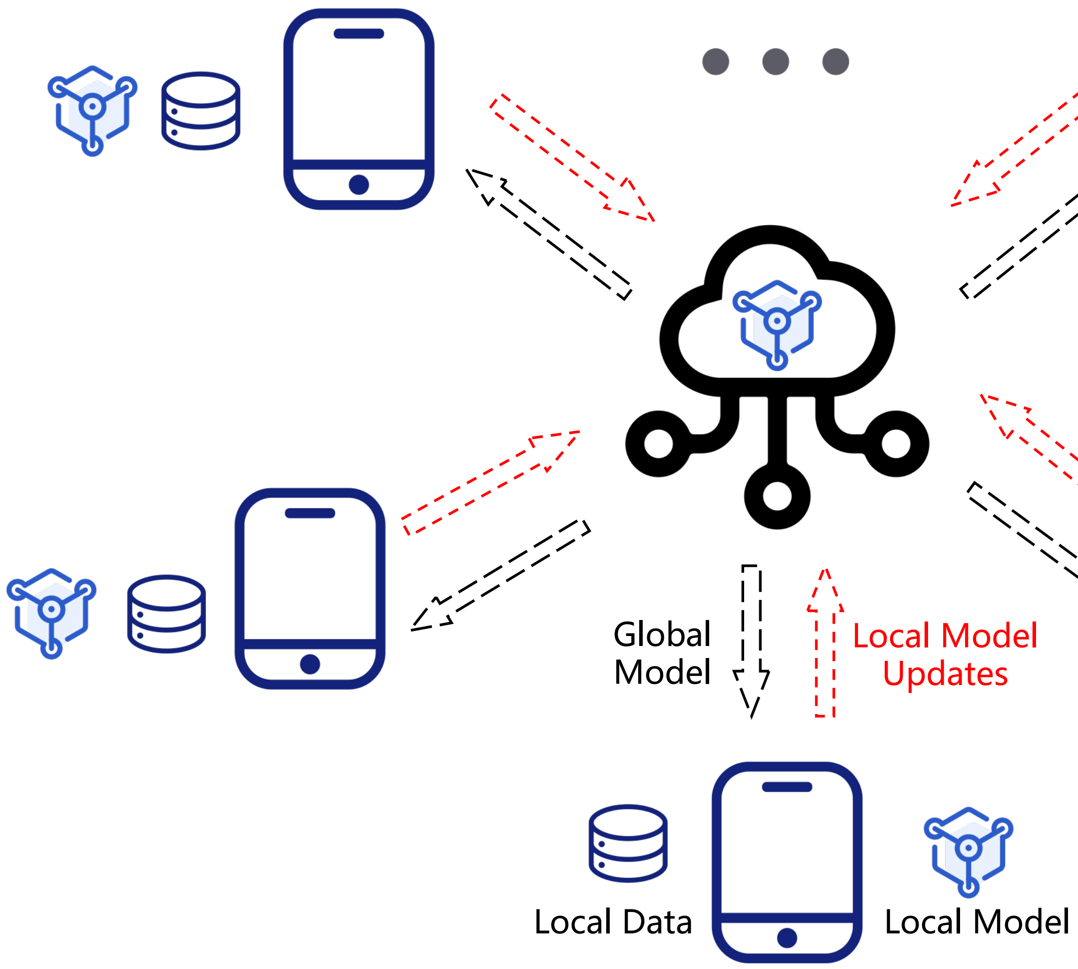
- Overview
- Personalized Federated Learning
- Federated Graph Learning
- Federated Hyperparameter Optimization
- Privacy Attacks

Personalized Federated Learning

Outline of PFL

- Why Personalized Federated Learninig (PFL)
- Existing PFL Methods
- PFL Hands-On Practice
- PFL Benchmark

Why PFL? Next-Word Prediction Case



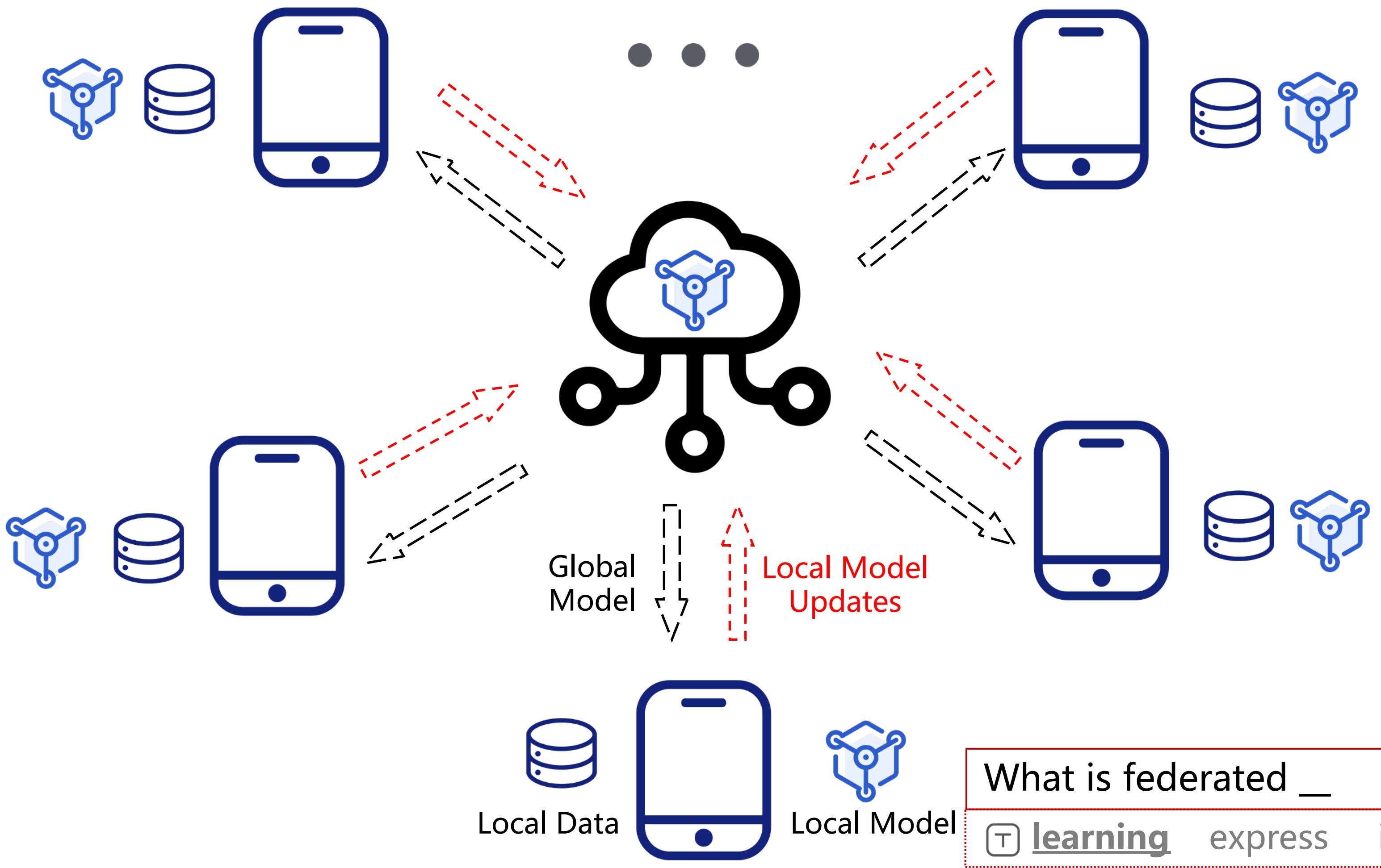
What is federated __

learning express insurance

Typical Flow

1. Server selects clients, sends msg
2. Clients learn on private data
3. Clients upload msg (model)
4. Server aggregates msgs

Ideally: IID data



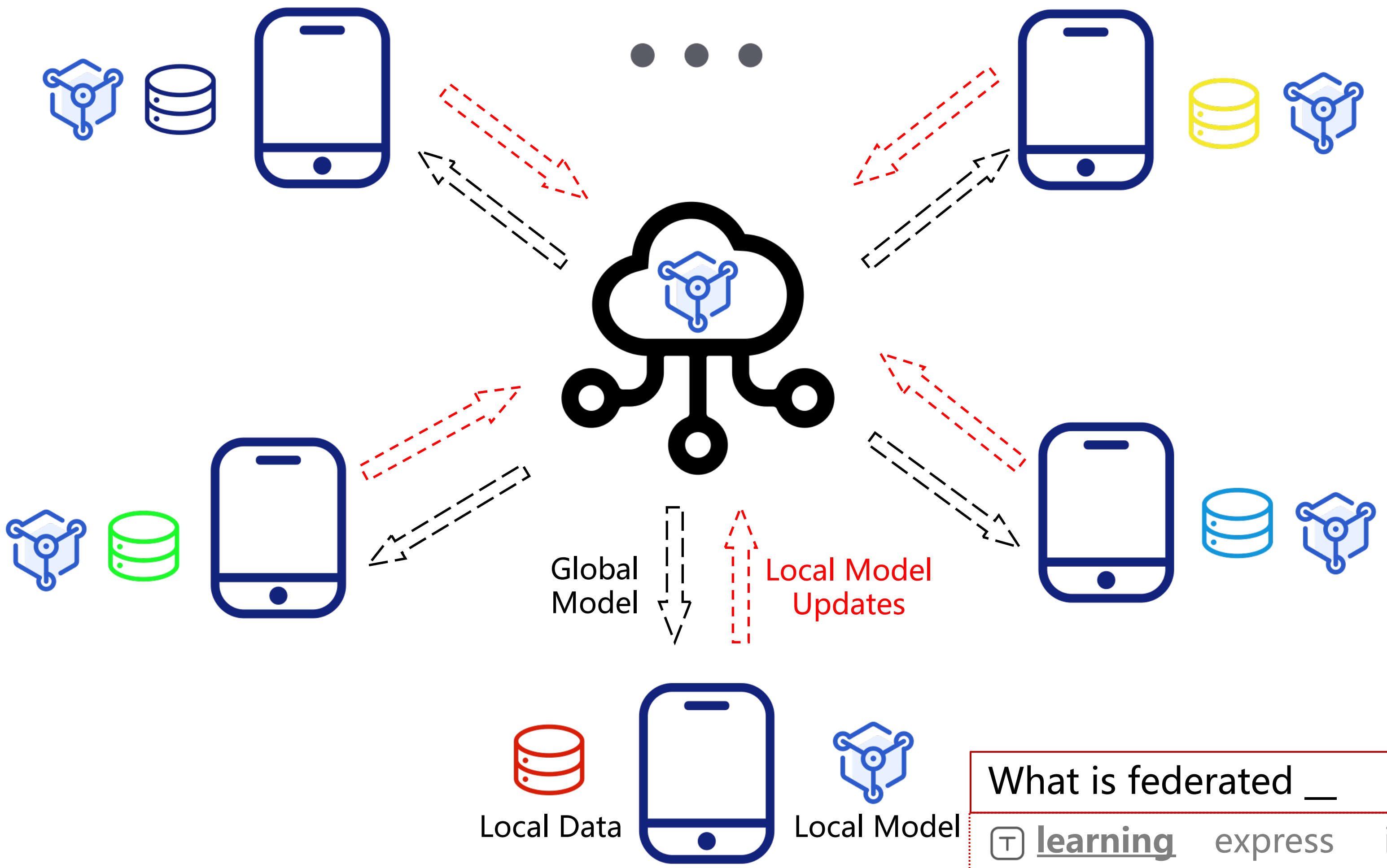
What is federated __

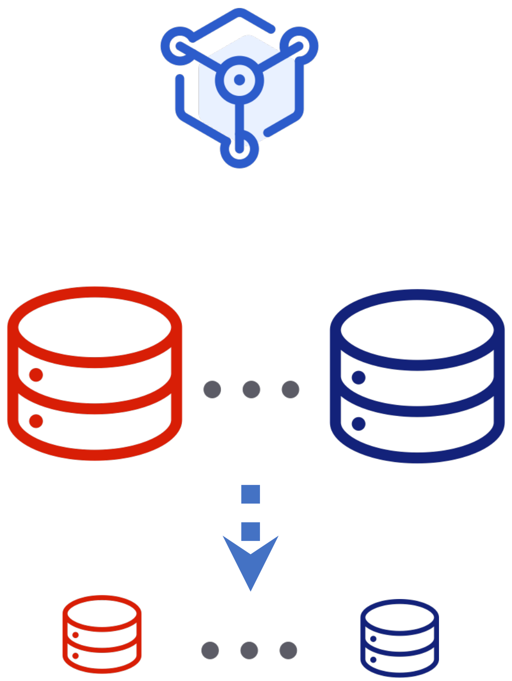
learning express insurance

Global model: one-for-all

good-for-all

Actually: Non-IID Data

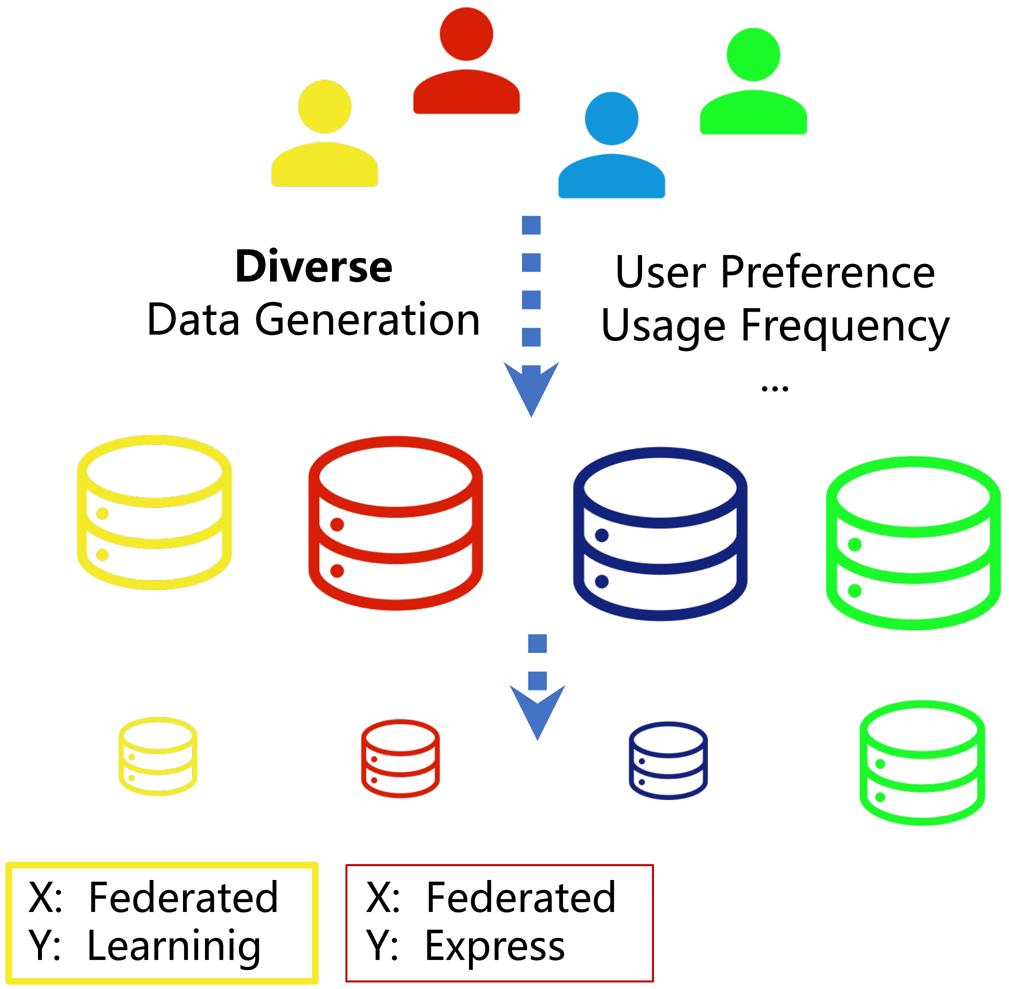




What is federated __

learning express insurance

Non-IID Data



Non-IID

- ➢ Marginal Distribution Skew
  - common P(Y|X); different P(X)
  - P(X|Y); P(Y)

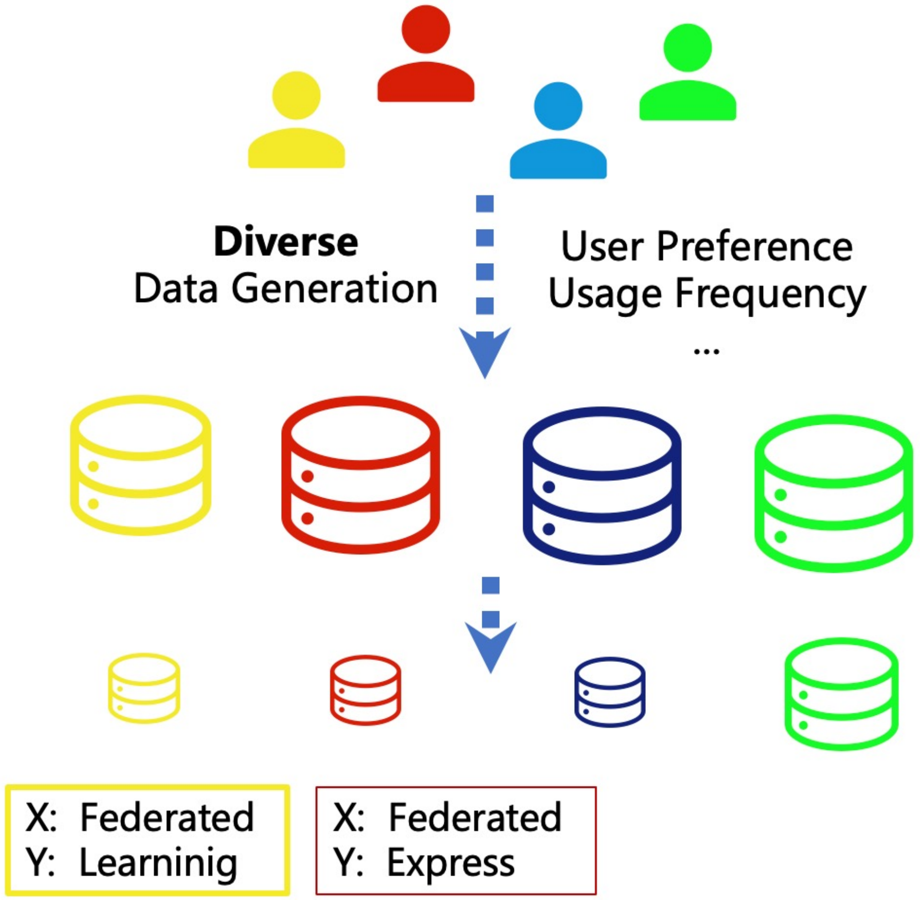
  e.g., Users in different regions differ in their vocabularies

- ➢ Conditional Distribution Skew
  - common P(Y); different P(X|Y)
  - P(X); P(Y|X)

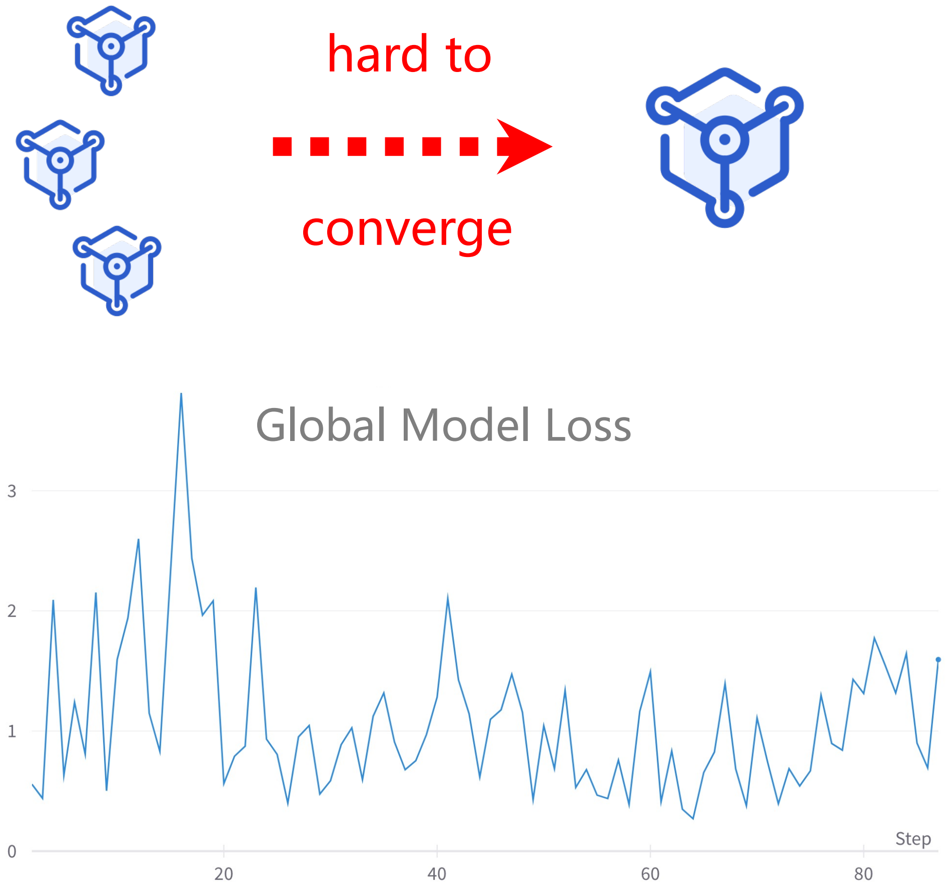
  e.g., Users expect different next words given the same input

- ➢ Quantity (i.e., size of local data) Skew

Non-IID Data

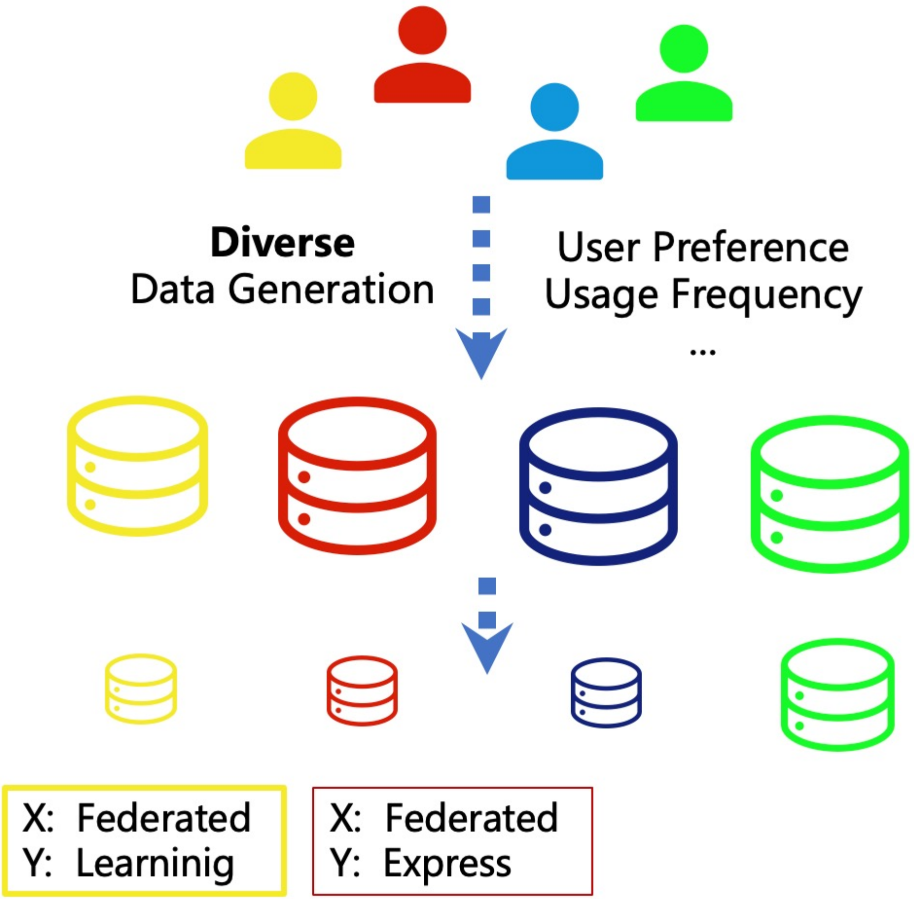


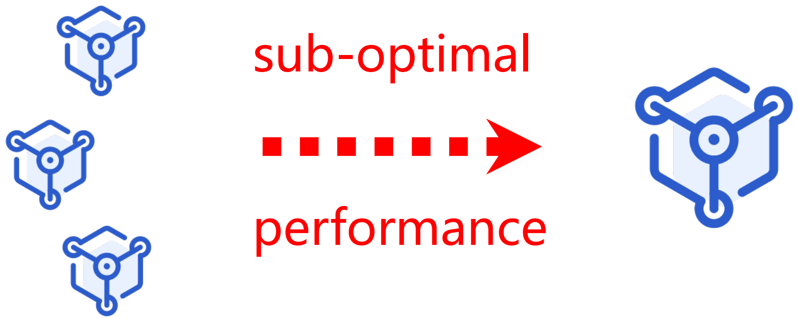
Diverse
Data Generation
User Preference
Usage Frequency
...
X: Federated
Y: Learninig
X: Federated
Y: Express

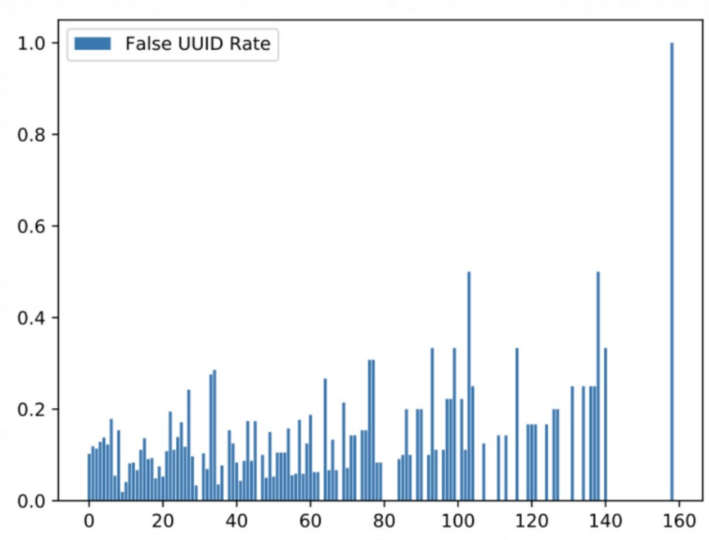


hard to
converge
Global Model Loss
Step

Non-IID Data

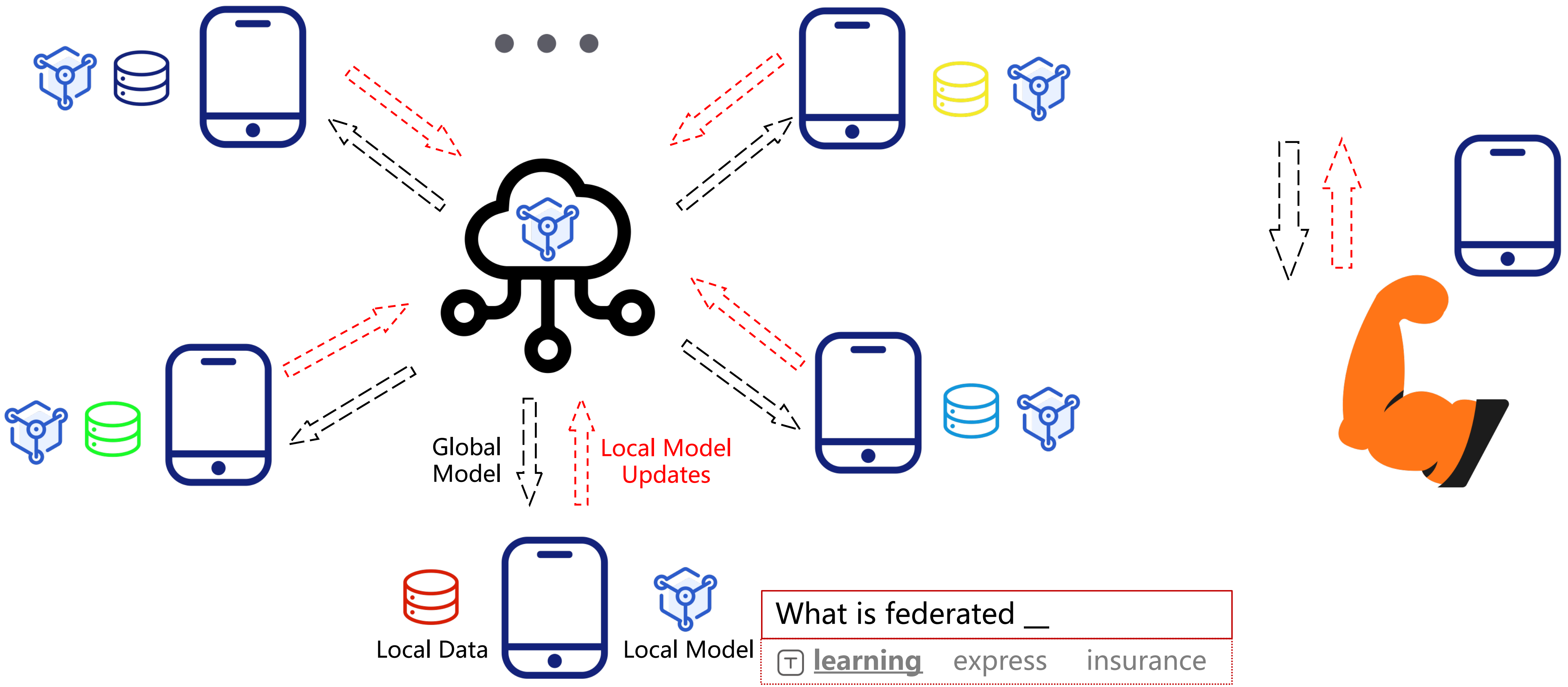




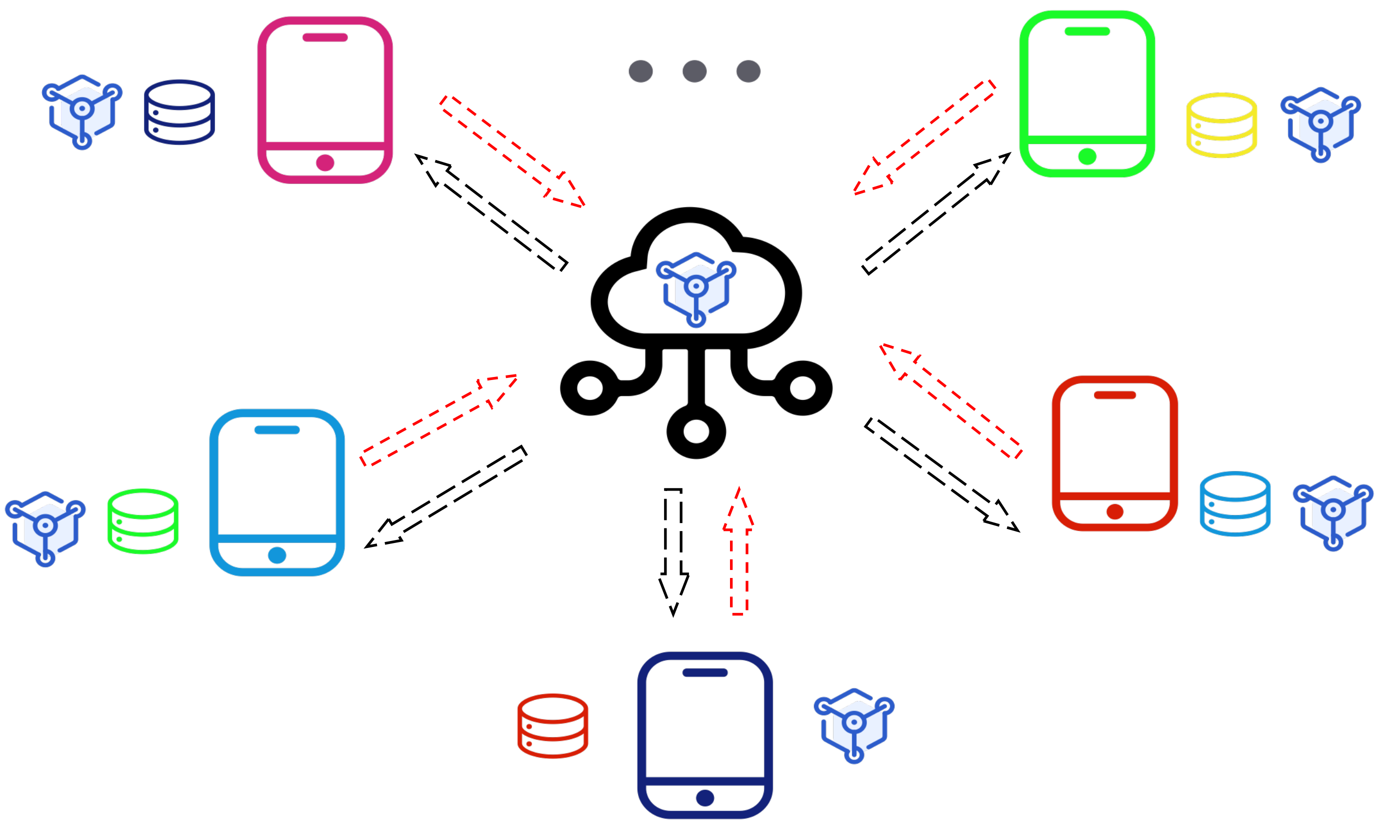


Client-wise performance

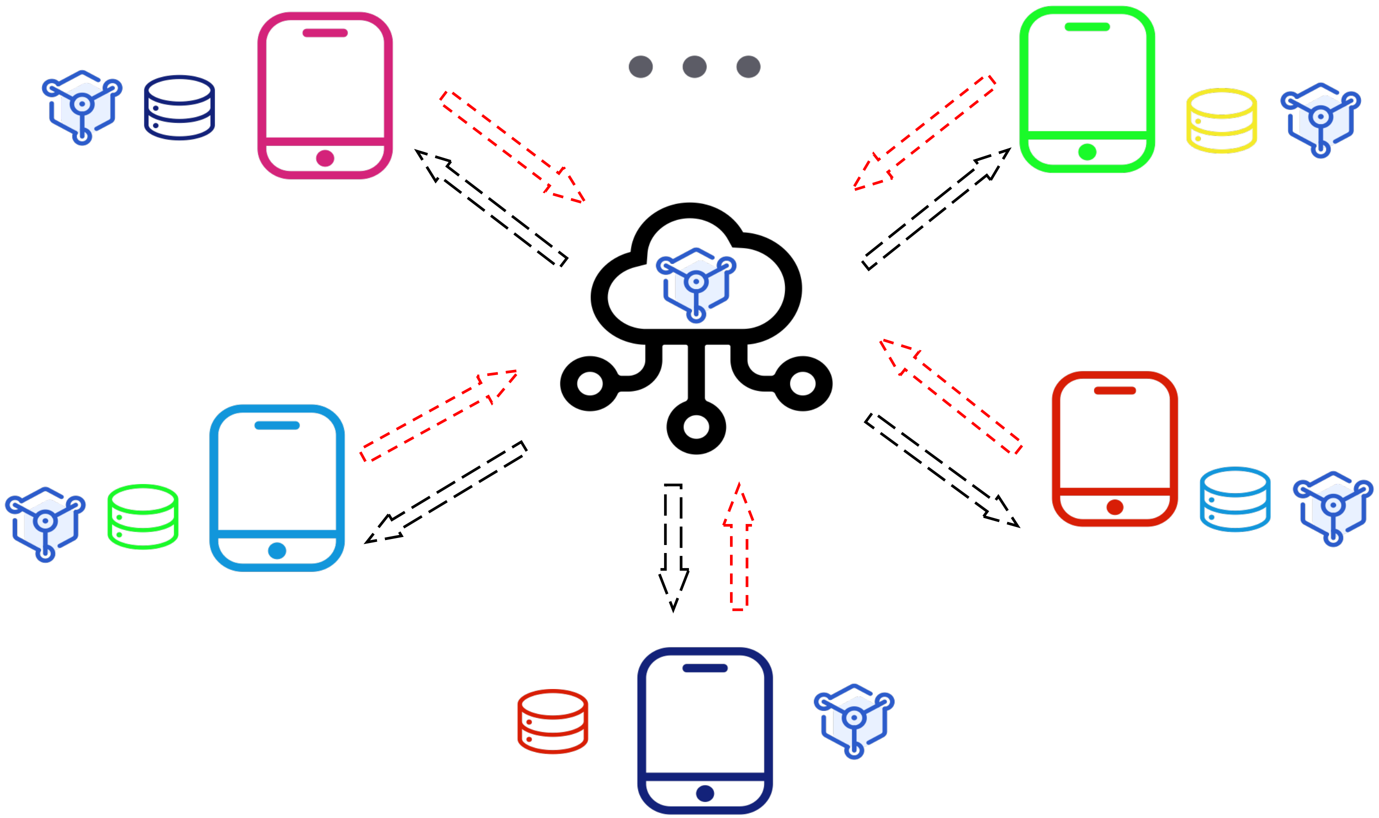
Ideally: Powerful System Capacity



Actually: Heterogeneous Capacities



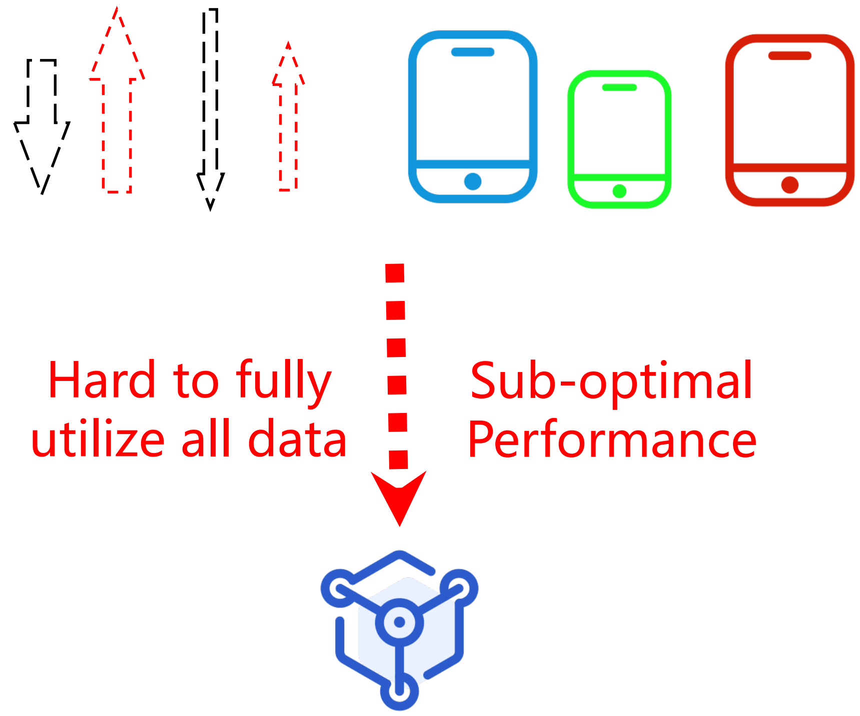
Actually: Heterogeneous Capacities



Diverse

Communication Bandwidth、QoS

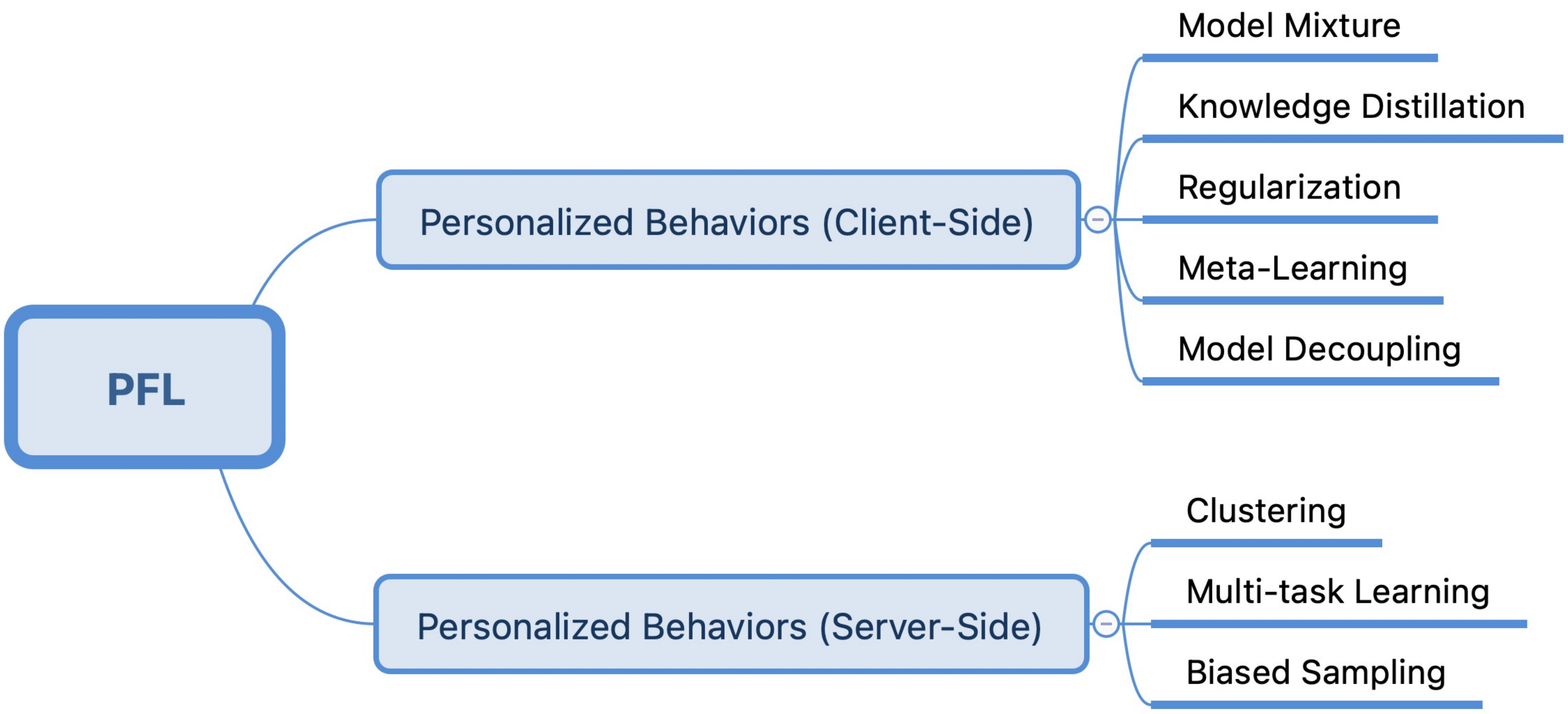
Computation Power、Storage



Outline

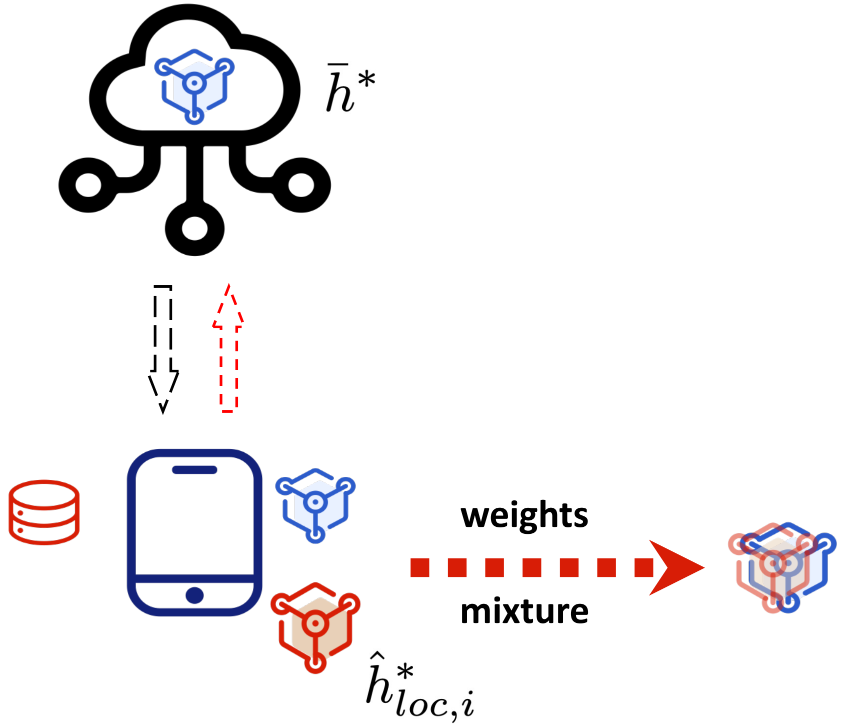
- Why Personalized Federated Learninig (PFL)
  - Non-IID Data
  - Heterogeous Device Capacities
- Existing PFL Methods
- PFL Hands-On Practice
- PFL Benchmark

Existing PFL Methods



Personalized Behaviors

- Model Mixture
- Knowledge Distillation
- Regularization
- Meta-Learning
- Model Decoupling



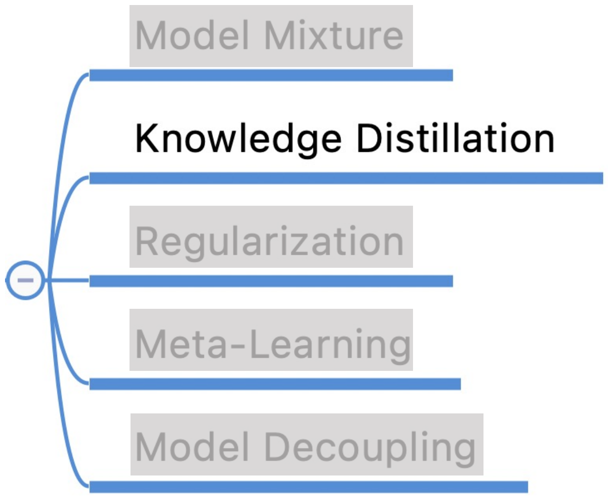
➢ pros: explicitly model local-global relationship; easy-to-implement

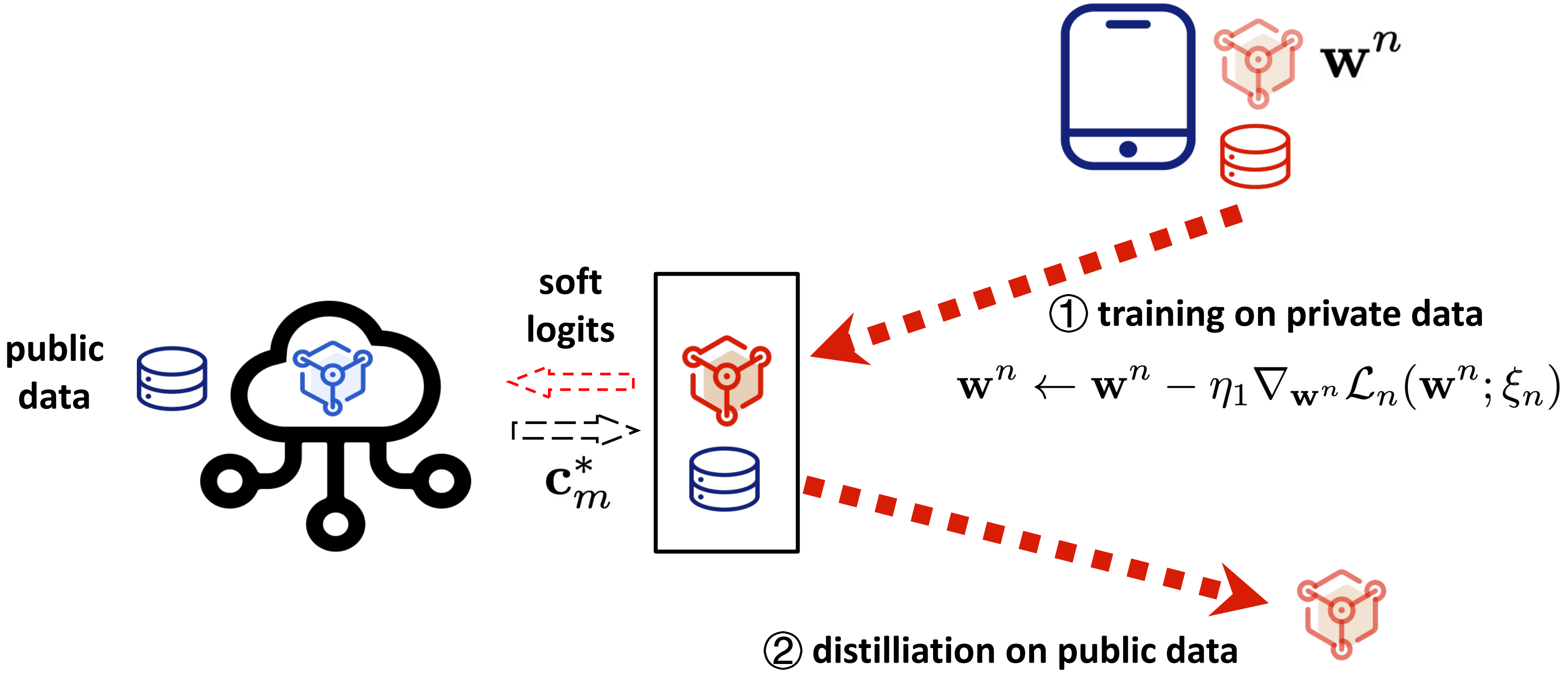
➢ cons: prune to be affected by single global model

$$\hat{h}^*_{loc,i} = \arg\min_{h \in \mathcal{H}} \hat{\mathcal{L}}_{\mathcal{D}_i}(\alpha_i h + (1-\alpha_i)\bar{h}^*)$$

[1] Adaptive personalized federated learning. In arXiv 2020.
[2] Personalized federated learning with first order model optimization. In ICLR 2020.

Personalized Behaviors



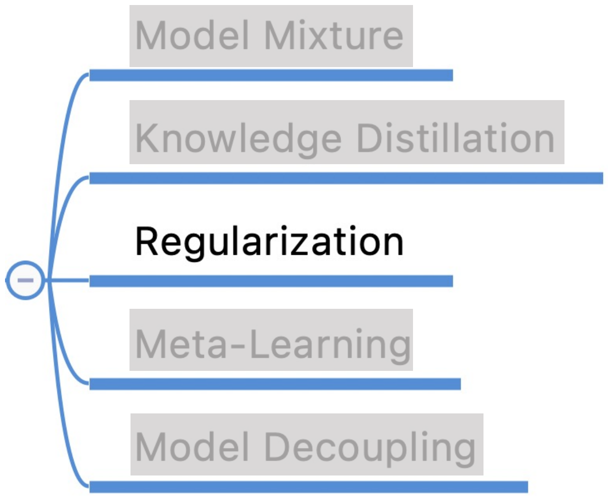


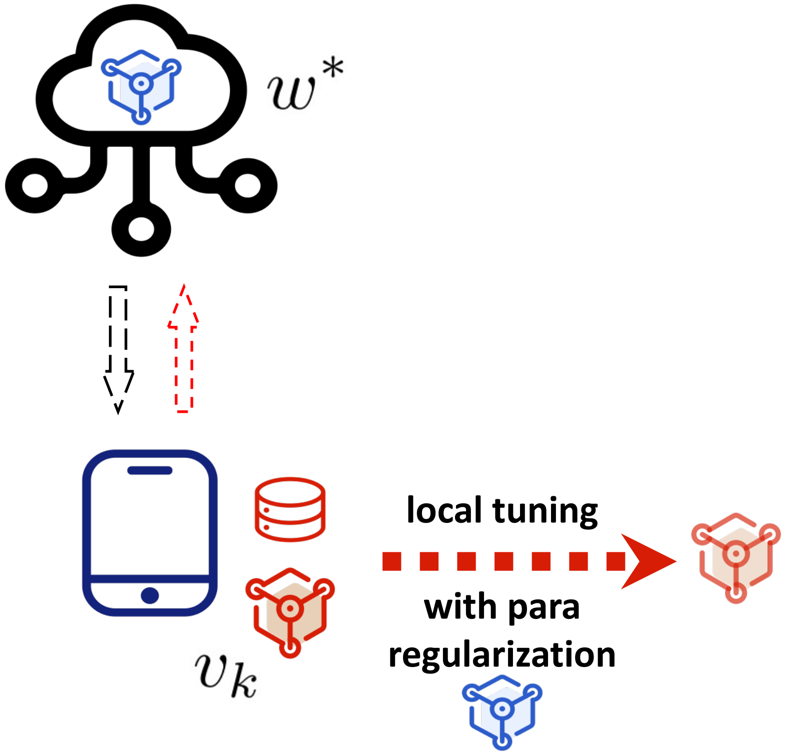
$$\mathbf{w}^n \leftarrow \mathbf{w}^n - \eta_2 \nabla_{\mathbf{w}^n} \mathcal{L}_{KL}\left(\sum_{m=1}^{N} \mathbf{c}_m^{*,T} \cdot s(\mathbf{w}^m, \xi_r), s(\mathbf{w}^n, \xi_r)\right)$$

- pros: low costs;
  flexible model architectures
- cons: may require public proxy data;
  hard to find optimal ensemble teacher model

[3] Parameterized Knowledge Transfer for Personalized Federated Learning. In NeurIPS 2021.
[4] Data-free knowledge distillation for heterogeneous federated learning. In ICML 2021.

Personalized Behaviors



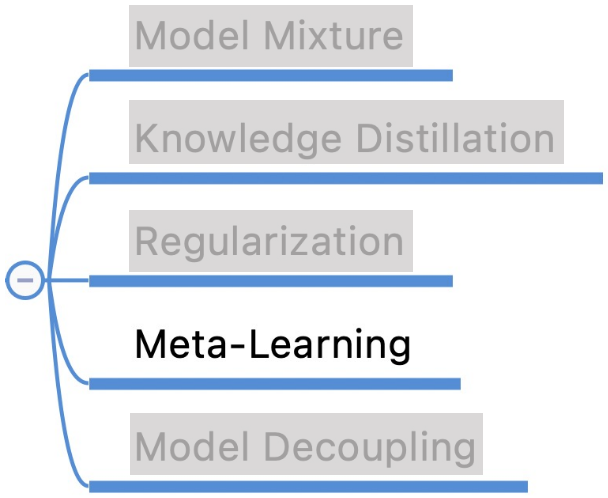


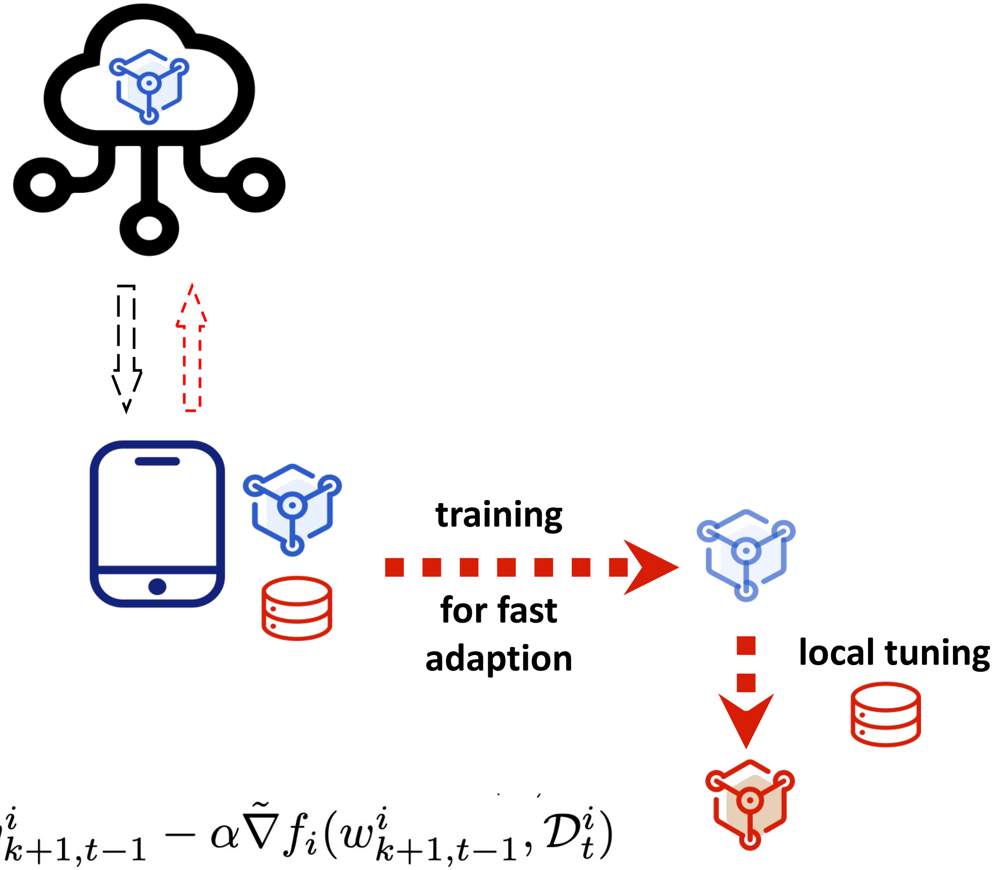
- pros: easy-to-implement;
  rich theoretical analysis
- cons: prune to be affected by the sinlge global model

$$\min_{v_k} \quad h_k(v_k; w^*) := F_k(v_k) + \frac{\lambda}{2} \|v_k - w^*\|^2$$
$$\text{s.t.} \quad w^* \in \arg\min_{w} G(F_1(w), \ldots F_K(w))\,.$$

[5] Ditto: Fair and robust federated learning through personalization. In ICML 2021.
[6] Personalized federated learning with moreau envelopes. In NeurIPS 2020.

Personalized Behaviors





- pros: benefit long-tailed clients
- cons: high storage & computing costs

$$\tilde{w}^i_{k+1,t} = w^i_{k+1,t-1} - \alpha\tilde{\nabla} f_i(w^i_{k+1,t-1}, \mathcal{D}^i_t)$$

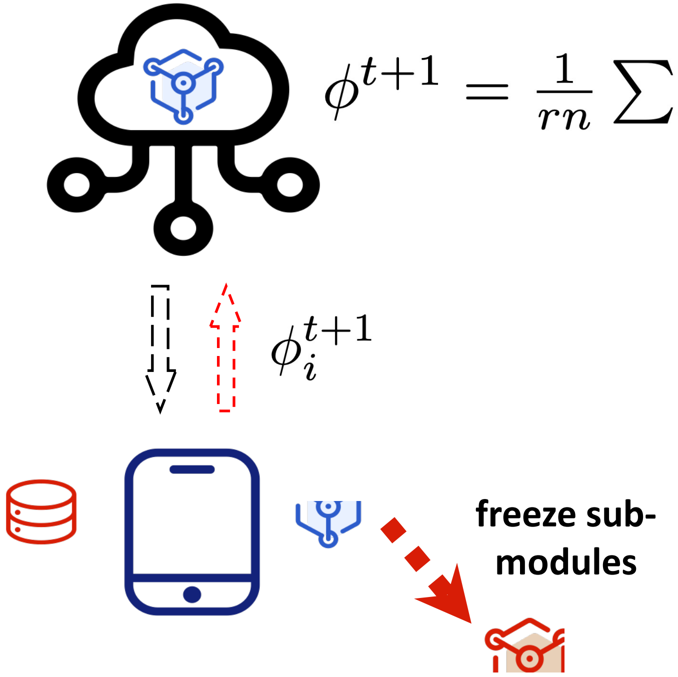
$$w^i_{k+1,t} = w^i_{k+1,t-1} - \beta(I - \alpha\tilde{\nabla}^2 f_i(w^i_{k+1,t-1}, \mathcal{D}''^i_t))\tilde{\nabla} f_i(\tilde{w}^i_{k+1,t}, \mathcal{D}'^i_t)$$

[6] Personalized federated learning with moreau envelopes. In NeurIPS 2020.
[7] Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In NeurIPS 2020.

Personalized Behaviors

- Model Mixture
- Knowledge Distillation
- Regularization
- Meta-Learning
- **Model Decoupling**

$$\phi^{t+1} = \frac{1}{rn} \sum_{i \in \mathcal{I}^t} \phi_i^{t+1}$$

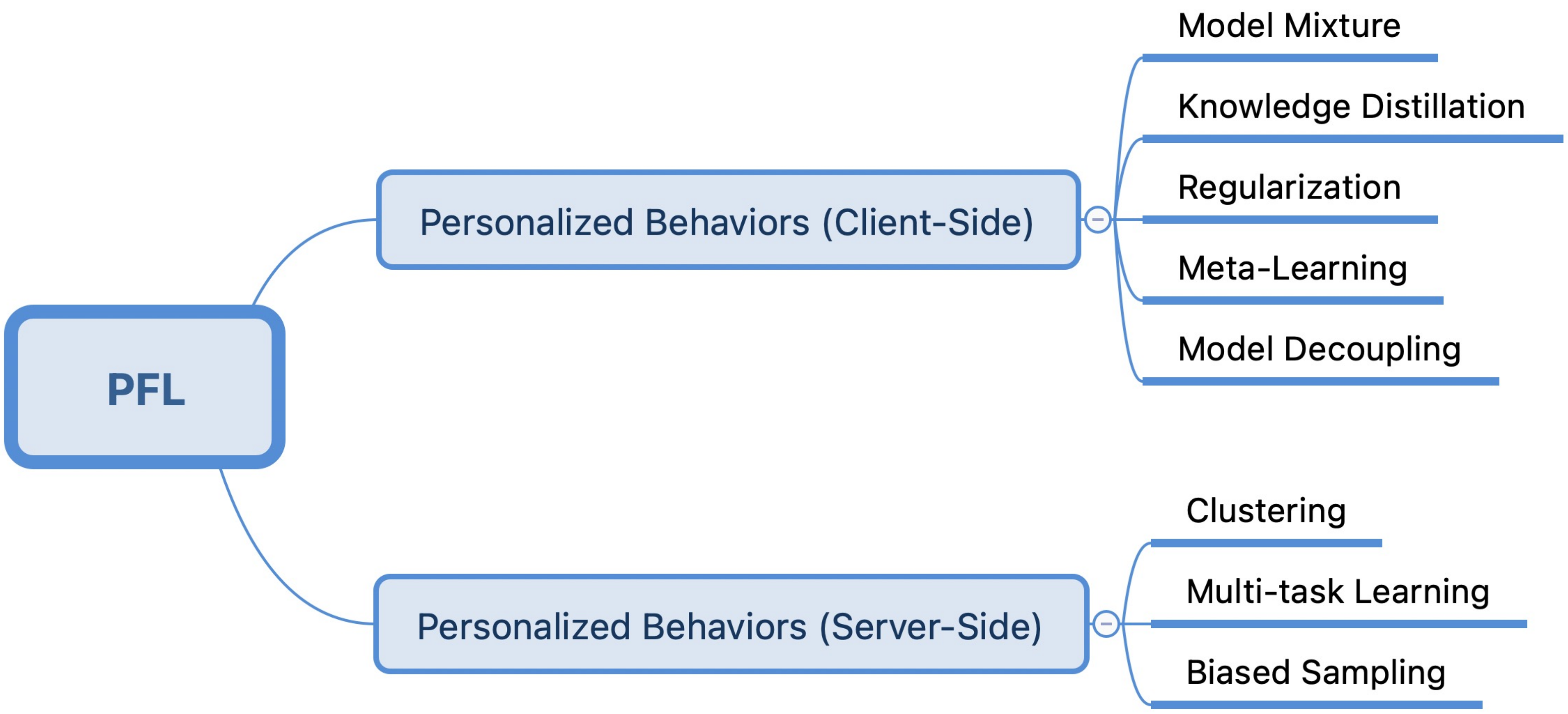


$$h_i^{t,s} \leftarrow \mathrm{GRD}(f_i(h_i^{t,s}, \phi^t), h_i^{t,s}, \alpha)$$

- pros: fine-grained personalization; low costs
- cons: hard to find optimal sub-model partititon

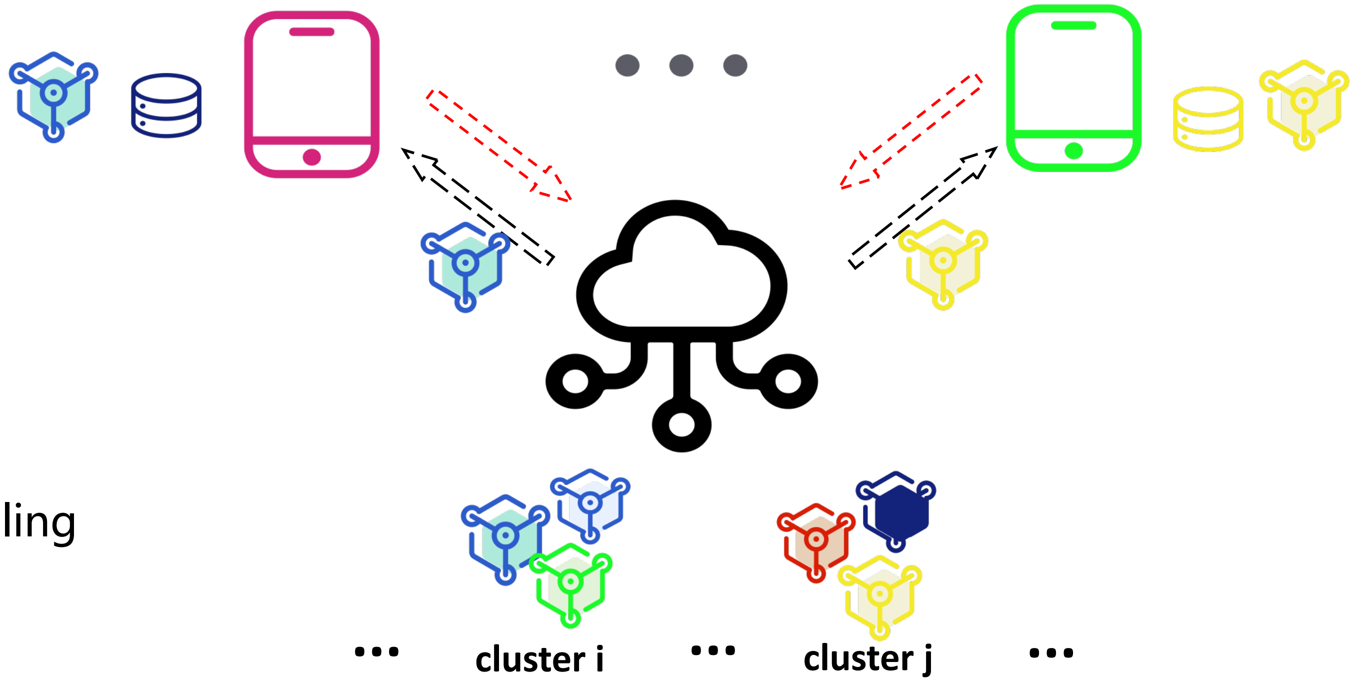
[8] FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. In ICLR 2021.
[9] Exploiting Shared Representations for Personalized Federated Learning. In ICML 2021

Existing PFL Methods



Personalized Behaviors

- Clustering
- Multi-task Learning
- Biased Sampling

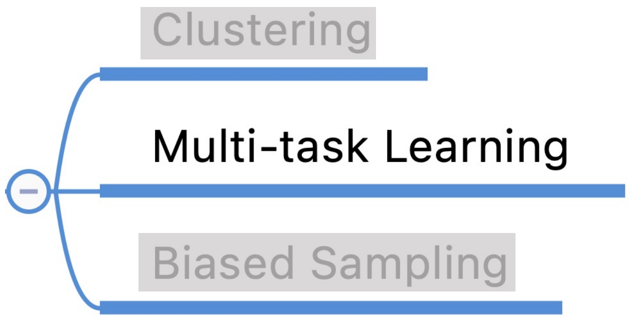


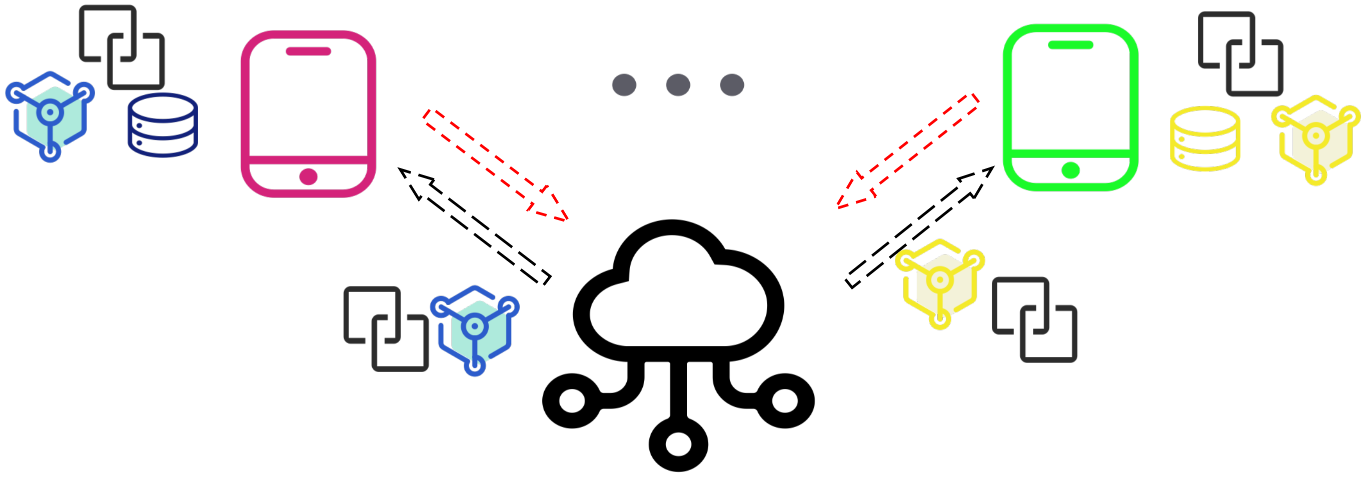
$$\theta_j^* = \text{argmin}_{\theta \in \Theta} F^j(\theta), j \in [k]$$

- ➢ pros: group-wise relationship modeling
- ➢ cons: high storage, computing, communication costs

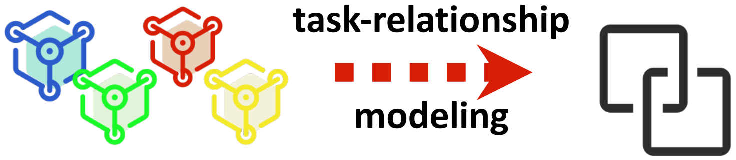
[10] An efficient framework for clustered federated learning. In NeurIPS 2020.
[11] Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. In TNNLS 2020

Personalized Behaviors





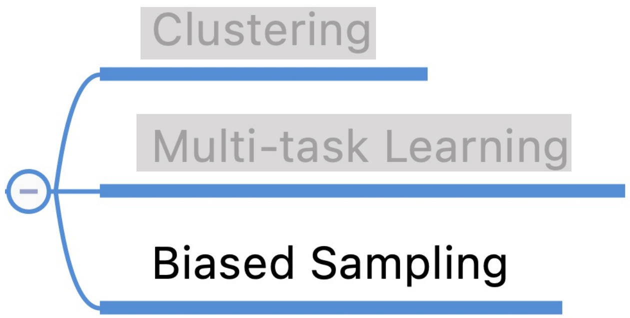
- pros: client-wise relationship modeling
- cons: high storage, computing, communication costs

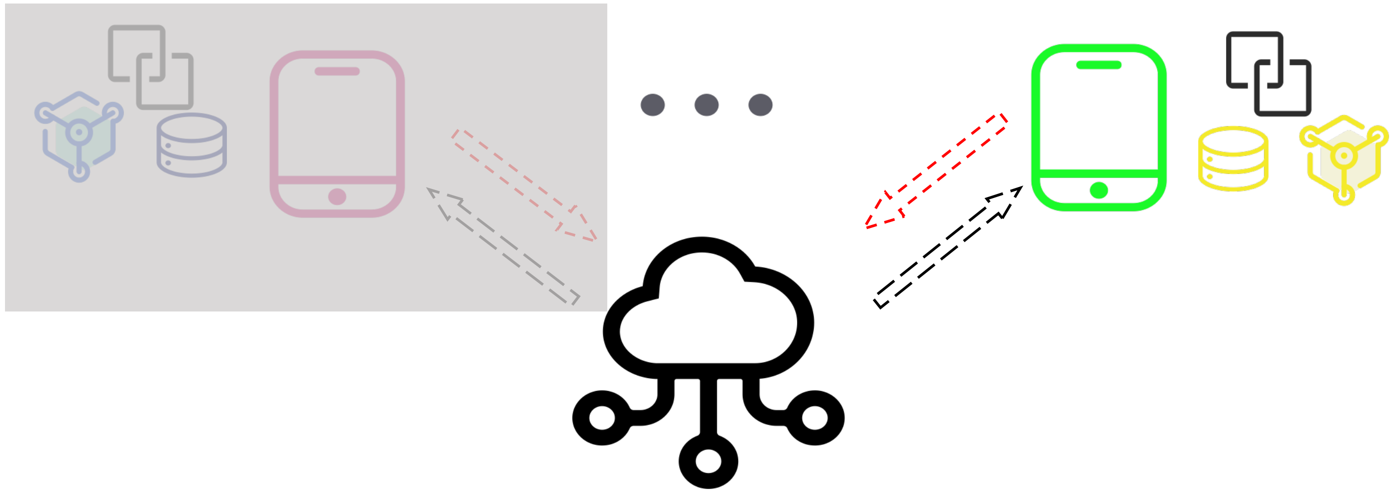


$$\min_{W,\Omega} \sum_{t=1}^{T} \sum_{i=1}^{n_t} l(h_{w_t}(\mathbf{x}_t^{(i)}), y_t^{(i)}) + \lambda \operatorname{tr}(W \Omega W^\intercal)$$

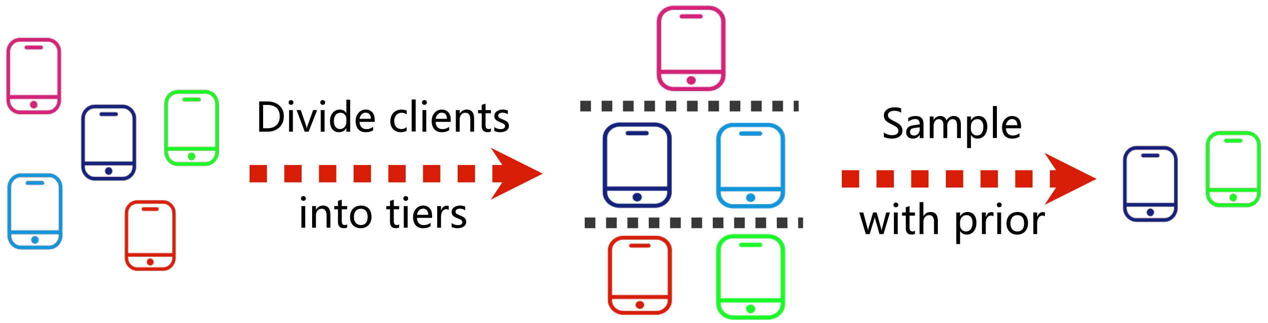
[12] Federated multi-task learning. In NeurIPS 2017.
[13] Federated multi-task learning under a mixture of distributions. In NeurIPS 2021.

Personalized Behaviors





- pros: easy-to-implement;
  improved convergence
- cons: may sacrifice long-tailed clients



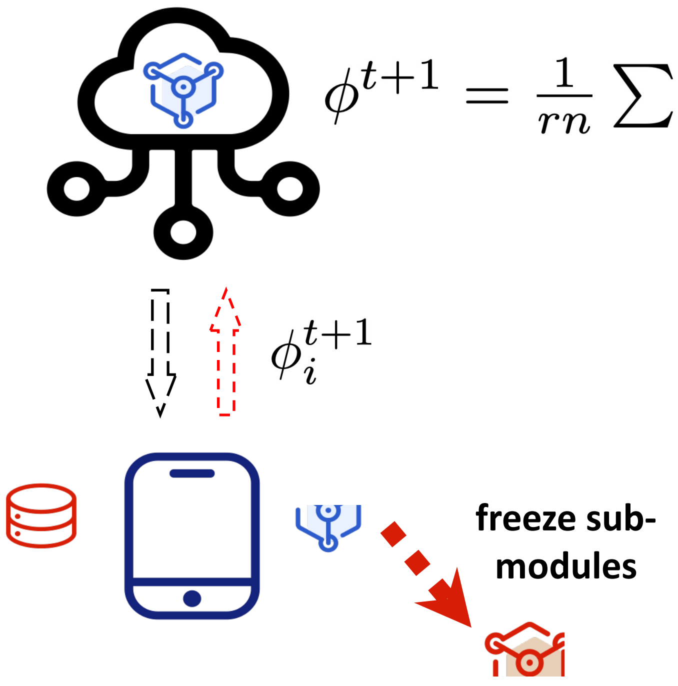
[14] Tifl: A tier-based federated learning system. In ACM HPDC 2020.

Outline

- Why Personalized Federated Learninig (PFL)
- Existing PFL Methods
- PFL Hands-On Practice
- PFL Benchmark

From FedAvg to FedBN

- FedBN [8]
  - Locally maintain model sub-parameters that *encode local data knowledge*
  - e.g. BN layers



$$\phi^{t+1} = \frac{1}{rn} \sum_{i \in \mathcal{I}^t} \phi_i^{t+1}$$

$$\phi_i^{t+1}$$

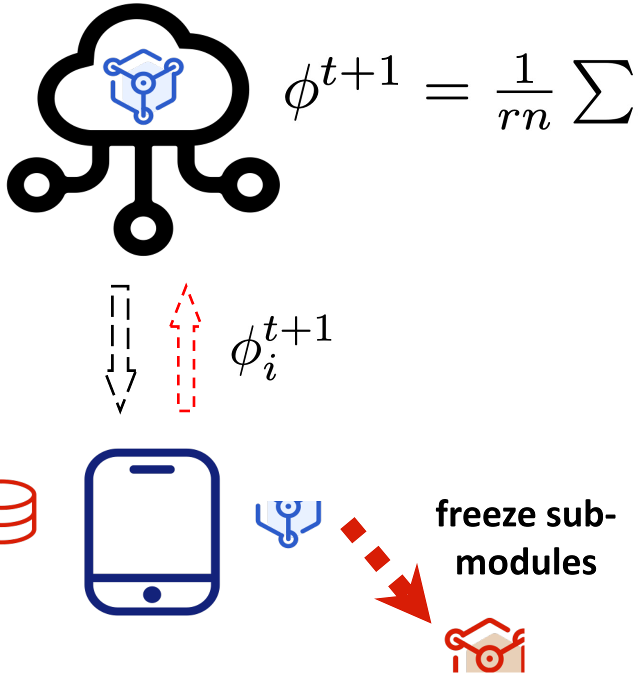
$$h_i^{t,s} \leftarrow \texttt{GRD}(f_i(h_i^{t,s}, \phi^t), h_i^{t,s}, \alpha)$$

[8] FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. In ICLR 2021.

From FedAvg to FedBN

- FedBN [8]
  - Locally maintain model sub-parameters that *encode local data knowledge*
  - e.g. BN layers
- We need to carefully filter out the model sub-parameters during FL courses
  - client local training
  - server aggregation

$$\phi^{t+1} = \frac{1}{rn} \sum_{i \in \mathcal{I}^t} \phi_i^{t+1}$$



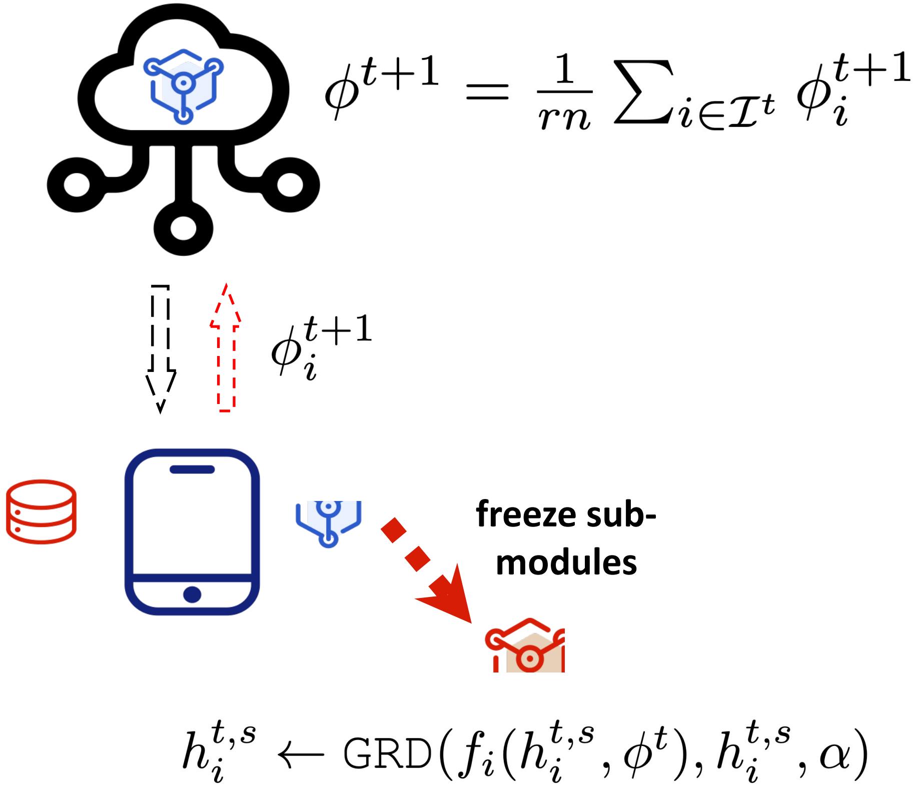
$$h_i^{t,s} \leftarrow \mathtt{GRD}(f_i(h_i^{t,s}, \phi^t), h_i^{t,s}, \alpha)$$

[8] FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. In ICLR 2021.

FedBN in FederatedScope

- Local model sub-parameter via FS
  - single-line **configuration**

```
cfg.personalization.local_param = []
# e.g., ['pre', 'post', 'bn']
```



[8] FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. In ICLR 2021.

FedBN in FederatedScope

- Local model sub-parameter via FS
  - single-line configuration

```
cfg.personalization.local_param = []
# e.g., ['pre', 'post', 'bn']
```

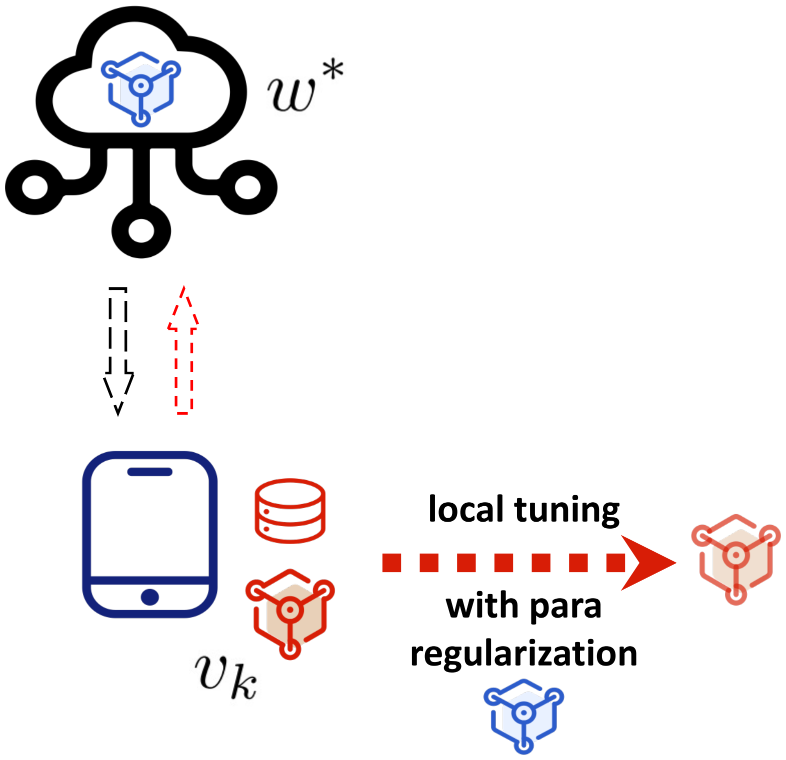
  - **auto filtering** during FL processes

```
# trainer.print_trainer_meta_info()

Model meta-info: <class 'federatedscope.cv.model.cnn.ConvNet2'>.
Num of original para names: 18.
Num of original trainable para names: 12.
Num of preserved para names in local update: 8.
Preserved para names in local update: {'fc2.bias', 'conv1.weight', 'conv2.weigh
Num of filtered para names in local update: 10.
Filtered para names in local update: {'bn2.weight', 'bn2.num_batches_tracked',
```

[8] FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. In ICLR 2021.

From FedAvg to Ditto

- Ditto [5]
  - maintain both local & global models --> improved fairness & robustness
  - local model is trained with para regularization



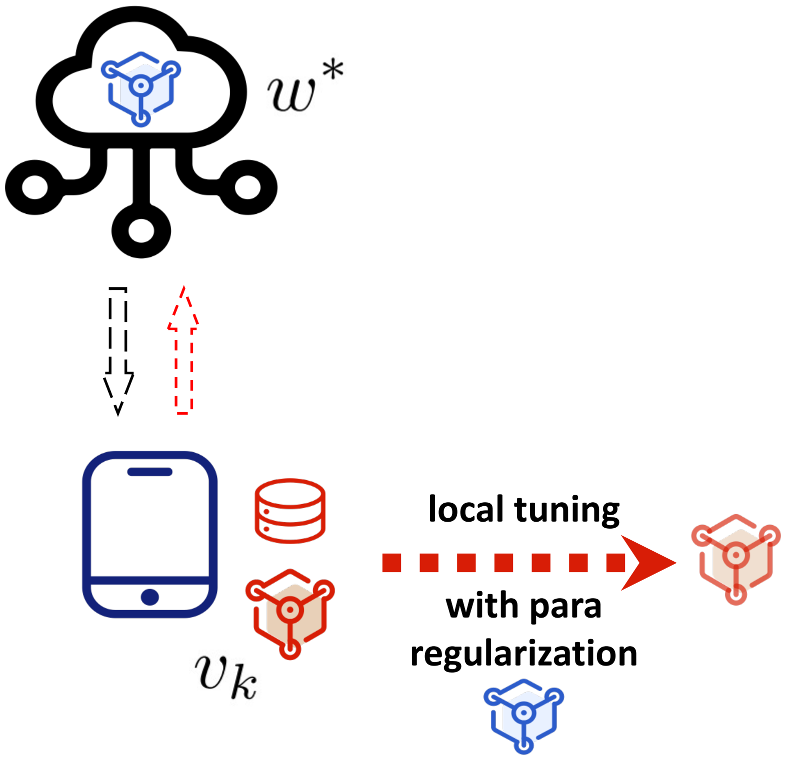
$$\min_{v_k} \quad h_k(v_k; w^*) := F_k(v_k) + \frac{\lambda}{2} \|v_k - w^*\|^2$$
$$\text{s.t.} \quad w^* \in \arg\min_w G(F_1(w), \ldots F_K(w)) .$$

[5] Ditto: Fair and robust federated learning through personalization. In ICML 2021.

From FedAvg to Ditto

- Ditto [5]
  - maintain both local & global models --> improved fairness & robustness
  - local model is trained with para regularization

- We need to
  - maintain client-specific model objects
  - (client) trains global model as FedAvg does
  - (client) trains local model with regularization



$$
\begin{aligned}
\min_{v_k} \quad & h_k(v_k; w^*) := F_k(v_k) + \frac{\lambda}{2} \|v_k - w^*\|^2 \\
\text{s.t.} \quad & w^* \in \arg\min_{w} G(F_1(w), \dots F_K(w)) \,.
\end{aligned}
$$

[5] Ditto: Fair and robust federated learning through personalization. In ICML 2021.

Ditto in FederatedScope

- Ditto attribute customization
  - maintain client-specific models

```
def wrap_DittoTrainer(
        base_trainer: Type[GeneralTorchTrainer]) -> Type[GeneralTorchTrainer]:

    # ---------------- attribute-level plug-in -----------------------
    init_Ditto_ctx(base_trainer)

    # ---------------- action-level plug-in -----------------------
    base_trainer.register_hook_in_train(
        new_hook=hook_on_fit_start_set_regularized_para,
        trigger="on_fit_start",
        insert_pos=0)
    ...
```

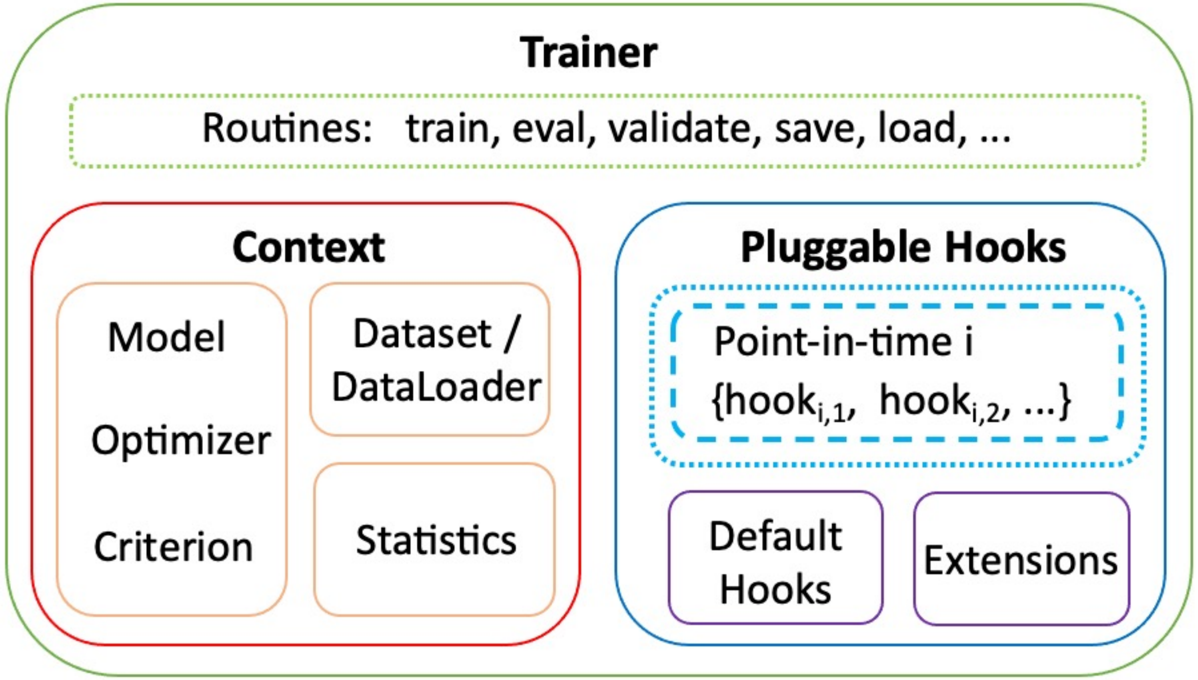
Ditto in FederatedScope

- Ditto behavior customization
  - incorporate parameter regularization into the local training

```
def wrap_DittoTrainer(
        base_trainer: Type[GeneralTorchTrainer]) -> Type[GeneralTorchTrainer]:

    # ---------------- attribute-level plug-in ----------------------
    init_Ditto_ctx(base_trainer)

    # ---------------- action-level plug-in ----------------------
    base_trainer.register_hook_in_train(
        new_hook=hook_on_fit_start_set_regularized_para,
        trigger="on_fit_start",
        insert_pos=0)
    ...
```

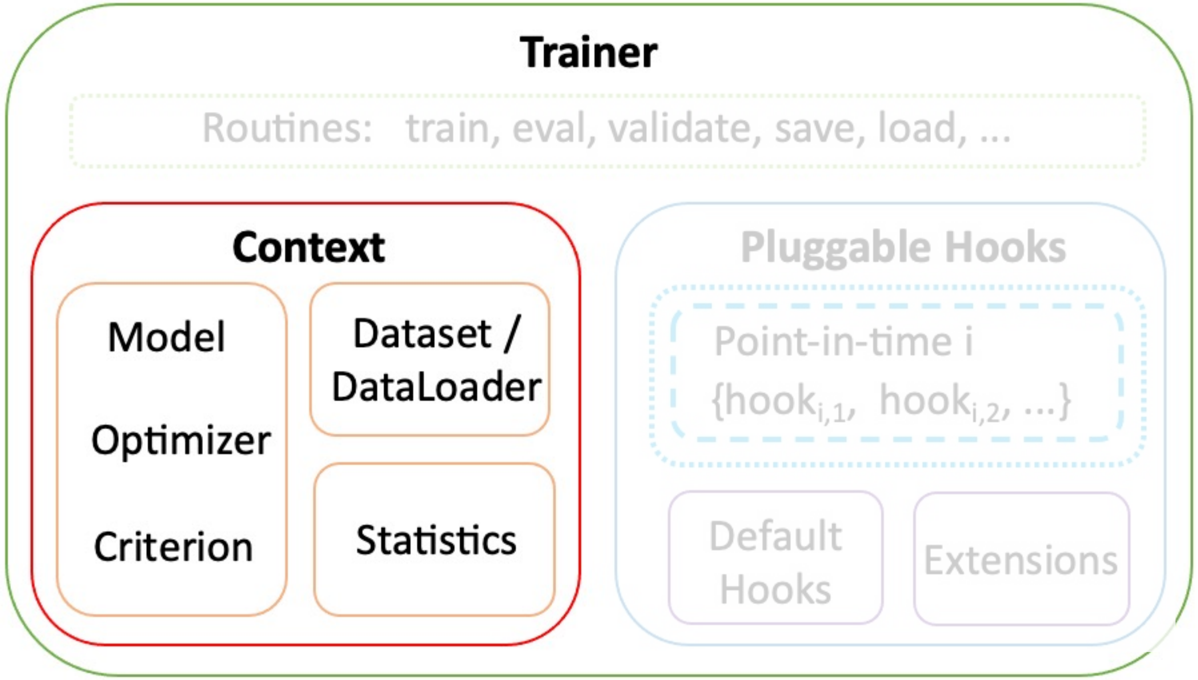
Customization with FS-Trainer

➢ Decoupled attributes & behaviors



Customization with FS-Trainer

- ➢ Decoupled attributes & behaviors
- ➢ Attribute Customization: *Context*
  - self-management life-cycle
  - built-in models, optimizers, etc.,



FS-Trainer-Context

- Context Variables
  - self-management life-cycle

```python
@lifecycle("routine")
def _run_routine(self, mode, dataset_name=None):
    for hook in self._get_hooks("on_fit_start"):
        hook(self.ctx)

    self._run_epoch()

    for hook in self._get_hooks("on_fit_end"):
        hook(self.ctx)
```

```python
@lifecycle("batch")
def _run_batch(self):
    batch_num = self.ctx.get(
        "num_{}_batch".format(self.ctx.cur_data_split))
    for batch_i in range(batch_num):
        self.ctx.cur_batch_i = CtxStatsVar(batch_i)

        for hook in self._get_hooks("on_batch_start"):
            hook(self.ctx)
        for hook in self._get_hooks("on_batch_forward"):
            hook(self.ctx)
        for hook in self._get_hooks("on_batch_backward"):
            hook(self.ctx)
        for hook in self._get_hooks("on_batch_end"):
            hook(self.ctx)

        # Break in the final epoch
        if self.ctx.cur_mode == 'train' and self.ctx.cur_epoch_i ==
                self.ctx.num_train_epoch - 1:
            if batch_i >= self.ctx.num_train_batch_last_epoch - 1:
                break
```

```python
@lifecycle("epoch")
def _run_epoch(self):
    epoch_num = self.ctx.get(
        "num_{}_epoch".format(self.ctx.cur_data_split))
    for epoch_i in range(epoch_num):
        self.ctx.cur_epoch_i = CtxStatsVar(epoch_i, "epoch")

        for hook in self._get_hooks("on_epoch_start"):
            hook(self.ctx)

        self._run_batch()

        for hook in self._get_hooks("on_epoch_end"):
            hook(self.ctx)
```

Implement Ditto via FS - Attribute

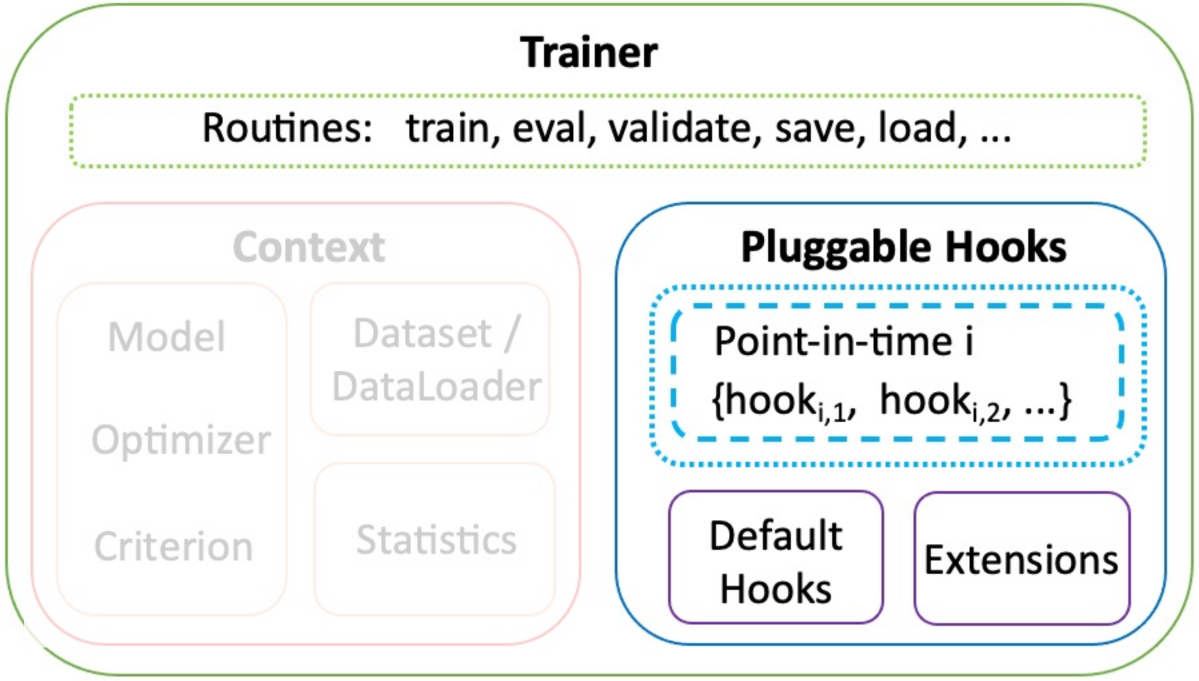
```
def wrap_DittoTrainer(
        base_trainer: Type[GeneralTorchTrainer]) -> Type[GeneralTorchTrainer]:

    # ---------------- attribute-level plug-in -----------------------
    init_Ditto_ctx(base_trainer)

    # ---------------- action-level plug-in -----------------------
    base_trainer.register_hook_in_train(
        new_hook=hook_on_fit_start_set_regularized_para,
        trigger="on_fit_start",
        insert_pos=0)
    ...
```

```
def init_Ditto_ctx(base_trainer):
    ctx.global_model = copy.deepcopy(ctx.model)
    ctx.local_model = copy.deepcopy(ctx.model)  # the personalized model
    ...
```

[5] Ditto: Fair and robust federated learning through personalization. In ICML 2021.

Customization with FS-Trainer

- Decoupled attributes & behaviors
- Attribute Customization: *Context*
  - self-management life-cycle
  - built-in models, optimizers, etc.,
- Behavior Customization
  - unified high-level routine APIs
  - pluggable hook functions



FS-Trainer-Hooks

- Point-in-time based pluggable hooks
  - {fit|epoch|batch}
  - {start|end|forward|backward}
- Hooks can be flexibly {insert, modify, replace}

```python
@lifecycle("routine")
def _run_routine(self, mode, dataset_name=None):
    for hook in self._get_hooks("on_fit_start"):
        hook(self.ctx)

    self._run_epoch()

    for hook in self._get_hooks("on_fit_end"):
        hook(self.ctx)
```

```python
@lifecycle("batch")
def _run_batch(self):
    batch_num = self.ctx.get(
        "num_{}_batch".format(self.ctx.cur_data_split))
    for batch_i in range(batch_num):
        self.ctx.cur_batch_i = CtxStatsVar(batch_i)

        for hook in self._get_hooks("on_batch_start"):
            hook(self.ctx)
        for hook in self._get_hooks("on_batch_forward"):
            hook(self.ctx)
        for hook in self._get_hooks("on_batch_backward"):
            hook(self.ctx)
        for hook in self._get_hooks("on_batch_end"):
            hook(self.ctx)

        # Break in the final epoch
        if self.ctx.cur_mode == 'train' and self.ctx.cur_epoch_i ==
                self.ctx.num_train_epoch - 1:
            if batch_i >= self.ctx.num_train_batch_last_epoch - 1:
                break
```

```python
@lifecycle("epoch")
def _run_epoch(self):
    epoch_num = self.ctx.get(
        "num_{}_epoch".format(self.ctx.cur_data_split))
    for epoch_i in range(epoch_num):
        self.ctx.cur_epoch_i = CtxStatsVar(epoch_i, "epoch")

        for hook in self._get_hooks("on_epoch_start"):
            hook(self.ctx)

        self._run_batch()

        for hook in self._get_hooks("on_epoch_end"):
            hook(self.ctx)
```

Implement Ditto via FS - Behavior

```
def wrap_DittoTrainer(
        base_trainer: Type[GeneralTorchTrainer]) -> Type[GeneralTorchTrainer]:

    # ---------------- attribute-level plug-in -----------------------
    init_Ditto_ctx(base_trainer)

    # ---------------- action-level plug-in ------------------------
    base_trainer.register_hook_in_train(
        new_hook=hook_on_fit_start_set_regularized_para,
        trigger="on_fit_start",
        insert_pos=0)
    ...
```

[5] Ditto: Fair and robust federated learning through personalization. In ICML 2021.

Implement Ditto via FS - Behavior

```python
def wrap_DittoTrainer(
        base_trainer: Type[GeneralTorchTrainer]) -> Type[GeneralTorchTrainer]:

    # ---------------- attribute-level plug-in -----------------------
    init_Ditto_ctx(base_trainer)

    # ---------------- action-level plug-in -----------------------
    base_trainer.register_hook_in_train(
        new_hook=hook_on_fit_start_set_regularized_para,
        trigger="on_fit_start",
        insert_pos=0)
    ...
```

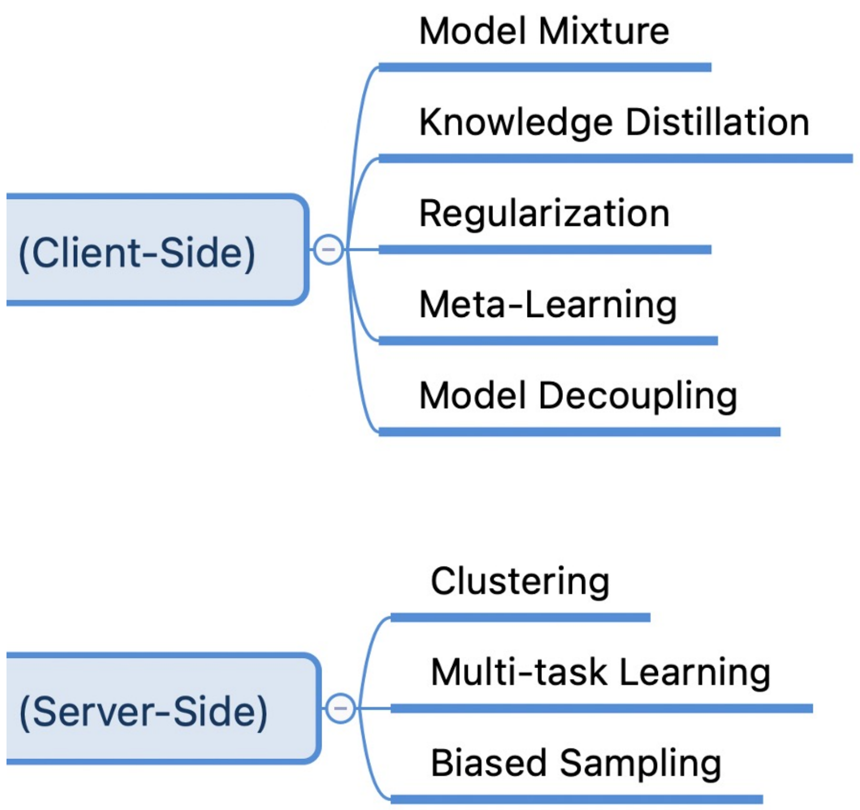
```python
def hook_on_fit_start_set_regularized_para(ctx):
    # After received the global model,
    # set the compared model data for local personalized model
    ctx.local_model.to(ctx.device)
    ctx.local_model.train()

    compared_global_model_para = [{
        "params": list(ctx.global_model.parameters())
    }]
    ctx.optimizer_for_local_model.set_compared_para_group(
        compared_global_model_para)
```

[5] Ditto: Fair and robust federated learning through personalization. In ICML 2021.

PFL Implementation via FS

FS provides flexible behavior customization

- Inter-Clients/Server Customization
  - event-driven
  - message/handler from own view
- Intra-Clients/Server Customization
  - modular *Trainer* object
  - clients/server distinct



Outline

- Why Personalized Federated Learninig (PFL)
- Existing PFL Methods
- PFL Hands-On Practice
- PFL Benchmark

PFL Benchmark – Implementations

ArXiv: [pFL-Bench: A Comprehensive Benchmark for Personalized Federated Learning]

- Fruitful pluggable hooks and sub-routines
- 20+ competitive PFL baseline implementations

Global-Train
Isolated
FedAvg
FedAvg-FT
FedOpt
FedOpt-FT
pFedMe
pFedMe-FT

FedEM
FedEM-FT
FedEM-FedBN
FedEM-FedBN-FT
FedEM-FedBN-FedOPT
FedEM-FedBN-FedOPT-FT

FedBN
FedBN-FT
FedBN-FedOPT
FedBN-FedOPT-FT
Ditto
Ditto-FT
Ditto-FedBN
Ditto-FedBN-FT
Ditto-FedBN-FedOpt
Ditto-FedBN-FedOpt-FT

Codes: https://github.com/alibaba/FederatedScope/tree/master/benchmark/pFL-Bench

PFL Benchmark – Datasets

Dataset	Task	Model	Partition By	# Clients	# Sample Per Client	
FEMNIST-5%	Image Classification	CNN	Writers	200	μ=217	σ=73
CIFAR10-α5			Labels	100	μ=600	σ=46
CIFAR10-α0.5				100	μ=600	σ=137
CIFAR10-α0.1				100	μ=600	σ=383
COLA	Linguistic Acceptability	BERT	Labels	50	μ=192	σ=159
SST-2	Sentiment Analysis			50	μ=1,364	σ=1,291
Cora	Node Classification	GIN	Community	5	μ=542	σ=30
Pubmed				5	μ=3,943	σ=34
Citeseer				5	μ=665	σ=29
MovieLens1M	Recommendation	MF	User	1000	μ=1,000	σ=482
MovieLens10M			Item	1000	μ=10,000	σ=8,155

[pFL-Bench: A Comprehensive Benchmark for Personalized Federated Learning]

PFL Benchmark – End2End Evaluation

- Server/Clients side monitoring


```
{
'Role': 'Server #',
'Round': 200,
'Results_weighted_avg': {
    'test_avg_loss': 0.58, 'test_acc': 0.67, 'test_correct': 3356,
    'test_loss': 2892, 'test_total': 5000
    },
'Results_avg': {
    'test_avg_loss': 0.57, 'test_acc': 0.67, 'test_correct': 3356,
    'test_loss': 2892, 'test_total': 5000
    },
'Results_fairness': {
    'test_correct': 3356,      'test_total': 5000,
    'test_avg_loss_std': 0.04, 'test_avg_loss_bottom_decile': 0.52,
    'test_avg_loss_top_decile': 0.64, 'test_acc_std': 0.06,
    'test_acc_bottom_decile': 0.60,      'test_acc_top_decile': 0.75,
    'test_loss_std': 214.17,    'test_loss_bottom_decile': 2644.64,
    'test_loss_top_decile': 3241.23
    },
}
```

PFL Benchmark – End2End Evaluation

- Server/Clients side monitoring
- Fruitful result aggregation manners
  - global avg
  - global weighted avg
  - individual


```
{
'Role': 'Server #',
'Round': 200,
'Results_weighted_avg': {
    'test_avg_loss': 0.58, 'test_acc': 0.67, 'test_correct': 3356,
    'test_loss': 2892, 'test_total': 5000
    },
'Results_avg': {
    'test_avg_loss': 0.57, 'test_acc': 0.67, 'test_correct': 3356,
    'test_loss': 2892, 'test_total': 5000
    },
'Results_fairness': {
    'test_correct': 3356,      'test_total': 5000,
    'test_avg_loss_std': 0.04, 'test_avg_loss_bottom_decile': 0.52,
    'test_avg_loss_top_decile': 0.64, 'test_acc_std': 0.06,
    'test_acc_bottom_decile': 0.60,      'test_acc_top_decile': 0.75,
    'test_loss_std': 214.17,    'test_loss_bottom_decile': 2644.64,
    'test_loss_top_decile': 3241.23
    },
}
```

PFL Benchmark – End2End Evaluation

- Server/Clients side monitoring
- Fruitful result aggregation manners
  - global avg
  - global weighted avg
  - individual
- Fruitful metrics
  - **generalization**: loss, acc, …
  - **fariness**: std, quantiles, …
  - **system efficiency**: flops, …


```
{
'Role': 'Server #',
'Round': 200,
'Results_weighted_avg': {
    'test_avg_loss': 0.58, 'test_acc': 0.67, 'test_correct': 3356,
    'test_loss': 2892, 'test_total': 5000
    },
'Results_avg': {
    'test_avg_loss': 0.57, 'test_acc': 0.67, 'test_correct': 3356,
    'test_loss': 2892, 'test_total': 5000
    },
'Results_fairness': {
    'test_correct': 3356,         'test_total': 5000,
    'test_avg_loss_std': 0.04, 'test_avg_loss_bottom_decile': 0.52,
    'test_avg_loss_top_decile': 0.64, 'test_acc_std': 0.06,
    'test_acc_bottom_decile': 0.60,      'test_acc_top_decile': 0.75,
    'test_loss_std': 214.17,    'test_loss_bottom_decile': 2644.64,
    'test_loss_top_decile': 3241.23
    },
}
```

PFL Benchmark – Generalization

Metric

$\overline{Acc}$: average weighted by local data size

$\widetilde{Acc}$: accuracy of un-participated clients

Δ : participation generalization gap

- **Bold** & <u>underlined</u>: best & second-best results among all methods
- Red & blue: best & second-best results for original methods w/o plug-ins "-"

	FEMNIST, $s = 0.2$		
	$\overline{Acc}$	$\widetilde{Acc}$	Δ
Global-Train	52.48	-	-
Isolated	68.74	-	-
FedAvg	83.97	81.97	-2.00
FedAvg-FT	86.44	84.94	<u>-1.50</u>
pFedMe	87.50	82.76	-4.73
pFedMe-FT	88.19	82.46	-5.73
FedBN	86.72	7.86	-78.86
FedBN-FT	88.51	82.87	-5.64
FedBN-FedOPT	88.25	8.77	-79.49
FedBN-FedOPT-FT	88.14	80.25	-7.88
Ditto	88.39	2.20	-86.19
Ditto-FT	85.72	56.96	-28.76
Ditto-FedBN	**88.94**	2.20	-86.74
Ditto-FedBN-FT	86.53	58.96	-27.57
Ditto-FedBN-FedOpt	<u>88.73</u>	2.20	-86.54
Ditto-FedBN-FedOpt-FT	87.02	55.22	-31.80
FedEM	84.35	82.81	-1.54
FedEM-FT	86.17	<u>85.01</u>	**-1.16**
FedEM-FedBN	84.37	12.88	-71.49
FedEM-FedBN-FT	88.29	83.96	-4.33
FedEM-FedBN-FedOPT	82.12	6.64	-75.48
FedEM-FedBN-FedOPT-FT	87.54	**85.76**	-1.79

PFL Benchmark – Generalization

Metric

$\overline{Acc}$: average weighted by local data size

$\widetilde{Acc}$: accuracy of un-participated clients

Δ : participation generalization gap

Original methods w/o plug-in "-"

- No dominant one
- Good intra-client generalization $\overline{Acc}$ with PFL

	FEMNIST, $s = 0.2$			SST-2			PUBMED		
	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ
Global-Train	52.48	-	-	**80.57**	-	-	87.01	-	-
Isolated	68.74	-	-	60.82	-	-	85.56	-	-
FedAvg	83.97	81.97	-2.00	74.88	80.24	5.36	87.27	72.63	-14.64
FedAvg-FT	86.44	84.94	-1.50	74.14	**83.28**	9.13	87.21	79.78	-7.43
pFedMe	87.50	82.76	-4.73	71.27	69.34	-1.92	86.91	71.64	-15.27
pFedMe-FT	88.19	82.46	-5.73	75.61	66.48	-9.13	85.71	77.07	-8.64
FedBN	86.72	7.86	-78.86	74.88	75.40	0.52	**88.49**	52.53	-35.95
FedBN-FT	88.51	82.87	-5.64	68.81	82.43	13.63	87.45	**80.36**	**-7.09**
FedBN-FedOPT	88.25	8.77	-79.49	64.70	65.50	0.81	87.87	42.72	-45.15
FedBN-FedOPT-FT	88.14	80.25	-7.88	68.65	70.56	1.91	87.54	77.07	-10.47
Ditto	88.39	2.20	-86.19	52.03	46.79	-5.24	87.27	2.84	-84.43
Ditto-FT	85.72	56.96	-28.76	56.49	65.50	9.01	87.47	35.03	-52.44
Ditto-FedBN	**88.94**	2.20	-86.74	56.03	46.79	-9.24	88.18	2.84	-85.34
Ditto-FedBN-FT	86.53	58.96	-27.57	53.15	66.49	13.34	87.83	28.52	-59.30
Ditto-FedBN-FedOpt	88.73	2.20	-86.54	57.67	46.79	-10.88	87.81	2.84	-84.97
Ditto-FedBN-FedOpt-FT	87.02	55.22	-31.80	52.89	66.49	13.60	87.60	18.18	-69.42
FedEM	84.35	82.81	-1.54	75.78	67.67	-8.11	85.64	71.12	-14.52
FedEM-FT	86.17	85.01	**-1.16**	64.86	81.63	**16.77**	85.88	78.08	-7.80
FedEM-FedBN	84.37	12.88	-71.49	75.43	62.81	-12.62	88.12	48.64	-39.48
FedEM-FedBN-FT	88.29	83.96	-4.33	64.96	81.04	**16.08**	86.38	72.02	-14.35
FedEM-FedBN-FedOPT	82.12	6.64	-75.48	72.25	64.69	-7.56	87.56	42.37	-45.19
FedEM-FedBN-FedOPT-FT	87.54	**85.76**	-1.79	62.26	73.87	11.61	87.49	72.39	-15.09

PFL Benchmark – Generalization

Metric

- $\overline{Acc}$: average weighted by local data size
- $\widetilde{Acc}$: accuracy of un-participated clients
- Δ : participation generalization gap

Huge potential in

- **PFL combination**

	FEMNIST, $s=0.2$			SST-2			PUBMED		
	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ
Global-Train	52.48	-	-	**80.57**	-	-	87.01	-	-
Isolated	68.74	-	-	60.82	-	-	85.56	-	-
FedAvg	83.97	81.97	-2.00	74.88	80.24	5.36	87.27	72.63	-14.64
FedAvg-FT	86.44	84.94	-1.50	74.14	**83.28**	9.13	87.21	79.78	-7.43
pFedMe	87.50	82.76	-4.73	71.27	69.34	-1.92	86.91	71.64	-15.27
pFedMe-FT	88.19	82.46	-5.73	75.61	66.48	-9.13	85.71	77.07	-8.64
FedBN	86.72	7.86	-78.86	74.88	75.40	0.52	**88.49**	52.53	-35.95
FedBN-FT	88.51	82.87	-5.64	68.81	82.43	13.63	87.45	**80.36**	**-7.09**
FedBN-FedOPT	88.25	8.77	-79.49	64.70	65.50	0.81	87.87	42.72	-45.15
FedBN-FedOPT-FT	88.14	80.25	-7.88	68.65	70.56	1.91	87.54	77.07	-10.47
Ditto	88.39	2.20	-86.19	52.03	46.79	-5.24	87.27	2.84	-84.43
Ditto-FT	85.72	56.96	-28.76	56.49	65.50	9.01	87.47	35.03	-52.44
Ditto-FedBN	**88.94**	2.20	-86.74	56.03	46.79	-9.24	88.18	2.84	-85.34
Ditto-FedBN-FT	86.53	58.96	-27.57	53.15	66.49	13.34	87.83	28.52	-59.30
Ditto-FedBN-FedOpt	88.73	2.20	-86.54	57.67	46.79	-10.88	87.81	2.84	-84.97
Ditto-FedBN-FedOpt-FT	87.02	55.22	-31.80	52.89	66.49	13.60	87.60	18.18	-69.42
FedEM	84.35	82.81	-1.54	75.78	67.67	-8.11	85.64	71.12	-14.52
FedEM-FT	86.17	85.01	**-1.16**	64.86	81.63	**16.77**	85.88	78.08	-7.80
FedEM-FedBN	84.37	12.88	-71.49	75.43	62.81	-12.62	88.12	48.64	-39.48
FedEM-FedBN-FT	88.29	83.96	-4.33	64.96	81.04	**16.08**	86.38	72.02	-14.35
FedEM-FedBN-FedOPT	82.12	6.64	-75.48	72.25	64.69	-7.56	87.56	42.37	-45.19
FedEM-FedBN-FedOPT-FT	87.54	**85.76**	-1.79	62.26	73.87	11.61	87.49	72.39	-15.09

PFL Benchmark – Generalization

Metric

- $\overline{Acc}$: average weighted by local data size
- $\widetilde{Acc}$: accuracy of un-participated clients
- Δ : participation generalization gap

Huge potential in

- PFL combination
- **inter-client generalization**

	FEMNIST, $s=0.2$			SST-2			PUBMED		
	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ
Global-Train	52.48	-	-	**80.57**	-	-	87.01	-	-
Isolated	68.74	-	-	60.82	-	-	85.56	-	-
FedAvg	83.97	81.97	-2.00	74.88	80.24	5.36	87.27	72.63	-14.64
FedAvg-FT	86.44	84.94	-1.50	74.14	**83.28**	9.13	87.21	79.78	-7.43
pFedMe	87.50	82.76	-4.73	71.27	69.34	-1.92	86.91	71.64	-15.27
pFedMe-FT	88.19	82.46	-5.73	75.61	66.48	-9.13	85.71	77.07	-8.64
FedBN	86.72	7.86	-78.86	74.88	75.40	0.52	**88.49**	52.53	-35.95
FedBN-FT	88.51	82.87	-5.64	68.81	82.43	13.63	87.45	**80.36**	**-7.09**
FedBN-FedOPT	88.25	8.77	-79.49	64.70	65.50	0.81	87.87	42.72	-45.15
FedBN-FedOPT-FT	88.14	80.25	-7.88	68.65	70.56	1.91	87.54	77.07	-10.47
Ditto	88.39	2.20	-86.19	52.03	46.79	-5.24	87.27	2.84	-84.43
Ditto-FT	85.72	56.96	-28.76	56.49	65.50	9.01	87.47	35.03	-52.44
Ditto-FedBN	**88.94**	2.20	-86.74	56.03	46.79	-9.24	88.18	2.84	-85.34
Ditto-FedBN-FT	86.53	58.96	-27.57	53.15	66.49	13.34	87.83	28.52	-59.30
Ditto-FedBN-FedOpt	88.73	2.20	-86.54	57.67	46.79	-10.88	87.81	2.84	-84.97
Ditto-FedBN-FedOpt-FT	87.02	55.22	-31.80	52.89	66.49	13.60	87.60	18.18	-69.42
FedEM	84.35	82.81	-1.54	75.78	67.67	-8.11	85.64	71.12	-14.52
FedEM-FT	86.17	85.01	**-1.16**	64.86	81.63	**16.77**	85.88	78.08	-7.80
FedEM-FedBN	84.37	12.88	-71.49	75.43	62.81	-12.62	88.12	48.64	-39.48
FedEM-FedBN-FT	88.29	83.96	-4.33	64.96	81.04	**16.08**	86.38	72.02	-14.35
FedEM-FedBN-FedOPT	82.12	6.64	-75.48	72.25	64.69	-7.56	87.56	42.37	-45.19
FedEM-FedBN-FedOPT-FT	87.54	**85.76**	-1.79	62.26	73.87	11.61	87.49	72.39	-15.09

PFL Benchmark – Generalization

Metric

$\overline{Acc}$: average weighted by local data size

$\widetilde{Acc}$: accuracy of un-participated clients

Δ : participation generalization gap

Huge potential in

- PFL combination
- inter-client generalization
- **text/graph datasets**

	FEMNIST, $s = 0.2$			SST-2			PUBMED		
	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ	$\overline{Acc}$	$\widetilde{Acc}$	Δ
Global-Train	52.48	-	-	**80.57**	-	-	87.01	-	-
Isolated	68.74	-	-	60.82	-	-	85.56	-	-
FedAvg	83.97	81.97	-2.00	74.88	80.24	5.36	87.27	72.63	-14.64
FedAvg-FT	86.44	84.94	-1.50	74.14	**83.28**	9.13	87.21	79.78	-7.43
pFedMe	87.50	82.76	-4.73	71.27	69.34	-1.92	86.91	71.64	-15.27
pFedMe-FT	88.19	82.46	-5.73	75.61	66.48	-9.13	85.71	77.07	-8.64
FedBN	86.72	7.86	-78.86	74.88	75.40	0.52	**88.49**	52.53	-35.95
FedBN-FT	88.51	82.87	-5.64	68.81	82.43	13.63	87.45	**80.36**	**-7.09**
FedBN-FedOPT	88.25	8.77	-79.49	64.70	65.50	0.81	87.87	42.72	-45.15
FedBN-FedOPT-FT	88.14	80.25	-7.88	68.65	70.56	1.91	87.54	77.07	-10.47
Ditto	88.39	2.20	-86.19	52.03	46.79	-5.24	87.27	2.84	-84.43
Ditto-FT	85.72	56.96	-28.76	56.49	65.50	9.01	87.47	35.03	-52.44
Ditto-FedBN	**88.94**	2.20	-86.74	56.03	46.79	-9.24	88.18	2.84	-85.34
Ditto-FedBN-FT	86.53	58.96	-27.57	53.15	66.49	13.34	87.83	28.52	-59.30
Ditto-FedBN-FedOpt	88.73	2.20	-86.54	57.67	46.79	-10.88	87.81	2.84	-84.97
Ditto-FedBN-FedOpt-FT	87.02	55.22	-31.80	52.89	66.49	13.60	87.60	18.18	-69.42
FedEM	84.35	82.81	-1.54	75.78	67.67	-8.11	85.64	71.12	-14.52
FedEM-FT	86.17	85.01	**-1.16**	64.86	81.63	**16.77**	85.88	78.08	-7.80
FedEM-FedBN	84.37	12.88	-71.49	75.43	62.81	-12.62	88.12	48.64	-39.48
FedEM-FedBN-FT	88.29	83.96	-4.33	64.96	81.04	**16.08**	86.38	72.02	-14.35
FedEM-FedBN-FedOPT	82.12	6.64	-75.48	72.25	64.69	-7.56	87.56	42.37	-45.19
FedEM-FedBN-FedOPT-FT	87.54	**85.76**	-1.79	62.26	73.87	11.61	87.49	72.39	-15.09

PFL Benchmark – Fairness

Metric

$\overline{Acc}'$: equally-weighted average

σ : std of client-wise accuracy

$\overset{\smile}{Acc}$: bottom (90-th) accuracy over all clients

- **Bold** & underlined: best & second-best results among all methods
- Red & blue: best & second-best results for original methods w/o plug-ins "-"

	FEMNIST, $s = 0.2$		
	$\overline{Acc}'$	σ	$\overset{\smile}{Acc}$
Isolated	67.08	10.76	53.16
FedAvg	82.40	9.91	69.11
FedAvg-FT	85.17	8.69	72.34
pFedMe	86.50	8.52	75.00
pFedMe-FT	87.06	8.02	75.00
FedBN	85.38	8.19	74.26
FedBN-FT	87.65	**6.33**	**80.02**
FedBN-FedOPT	87.27	7.34	76.87
FedBN-FedOPT-FT	87.13	7.36	78.27
Ditto	87.18	7.52	78.23
Ditto-FT	84.30	8.16	73.95
Ditto-FedBN	**87.82**	7.19	77.78
Ditto-FedBN-FT	85.16	7.98	75.25
Ditto-FedBN-FedOpt	87.64	7.08	78.23
Ditto-FedBN-FedOpt-FT	85.71	7.91	75.81
FedEM	82.61	9.57	69.29
FedEM-FT	84.91	8.39	73.64
FedEM-FedBN	82.94	9.35	70.43
FedEM-FedBN-FT	87.09	9.24	76.36
FedEM-FedBN-FedOPT	80.48	11.02	64.84
FedEM-FedBN-FedOPT-FT	86.23	8.33	75.44

PFL Benchmark – Fairness

Metric

$\overline{Acc}'$: equally-weighted average

σ : std of client-wise accuracy

$\breve{Acc}$: bottom (90-th) accuracy over all clients

- Bias in existing evaluation
- Good bottom acc with PFL

	FEMNIST, $s = 0.2$			SST-2			PUBMED		
	$\overline{Acc}'$	σ	$\breve{Acc}$	$\overline{Acc}'$	σ	$\breve{Acc}$	$\overline{Acc}'$	σ	$\breve{Acc}$
Isolated	67.08	10.76	53.16	59.40	41.29	0.00	84.67	6.26	74.63
FedAvg	82.40	9.91	69.11	76.30	22.02	44.85	86.72	3.93	79.76
FedAvg-FT	85.17	8.69	72.34	75.36	27.67	31.08	86.71	3.86	80.57
pFedMe	86.50	8.52	75.00	65.08	26.59	27.75	86.35	4.43	78.76
pFedMe-FT	87.06	8.02	75.00	74.36	27.02	32.49	85.47	**3.06**	80.95
FedBN	85.38	8.19	74.26	76.30	22.02	44.85	87.97	3.42	81.77
FedBN-FT	87.65	**6.33**	**80.02**	68.50	26.83	29.17	87.02	3.47	80.13
FedBN-FedOPT	87.27	7.34	76.87	65.59	31.07	22.22	87.43	4.64	80.81
FedBN-FedOPT-FT	87.13	7.36	78.27	68.42	28.18	30.71	87.02	3.94	81.78
Ditto	87.18	7.52	78.23	49.94	40.81	0.00	86.85	3.98	80.44
Ditto-FT	84.30	8.16	73.95	54.34	39.26	0.00	87.10	3.52	80.46
Ditto-FedBN	**87.82**	7.19	77.78	49.44	41.80	0.00	87.75	3.70	81.82
Ditto-FedBN-FT	85.16	7.98	75.25	52.18	39.85	0.00	87.43	3.77	81.15
Ditto-FedBN-FedOpt	87.64	7.08	78.23	55.61	40.43	1.39	87.27	3.90	79.14
Ditto-FedBN-FedOpt-FT	85.71	7.91	75.81	53.16	34.75	9.72	87.10	3.79	80.93
FedEM	82.61	9.57	69.29	**76.53**	23.34	44.44	85.05	4.44	78.51
FedEM-FT	84.91	8.39	73.64	64.29	32.84	12.96	85.54	4.48	79.39
FedEM-FedBN	82.94	9.35	70.43	75.06	**18.48**	**53.33**	87.63	4.14	**82.54**
FedEM-FedBN-FT	87.09	9.24	76.36	64.33	35.72	8.59	85.68	4.33	79.44
FedEM-FedBN-FedOPT	80.48	11.02	64.84	72.66	27.18	34.17	87.11	4.24	80.32
FedEM-FedBN-FedOPT-FT	86.23	8.33	75.44	58.42	31.21	17.93	87.16	3.66	82.20

PFL Benchmark – Fairness

Metric

$\overline{Acc}'$: equally-weighted average

σ : std of client-wise accuracy

$\overset{\smile}{Acc}$: bottom (90-th) accuracy over all clients

- Huge potential in domain-specific fairness study

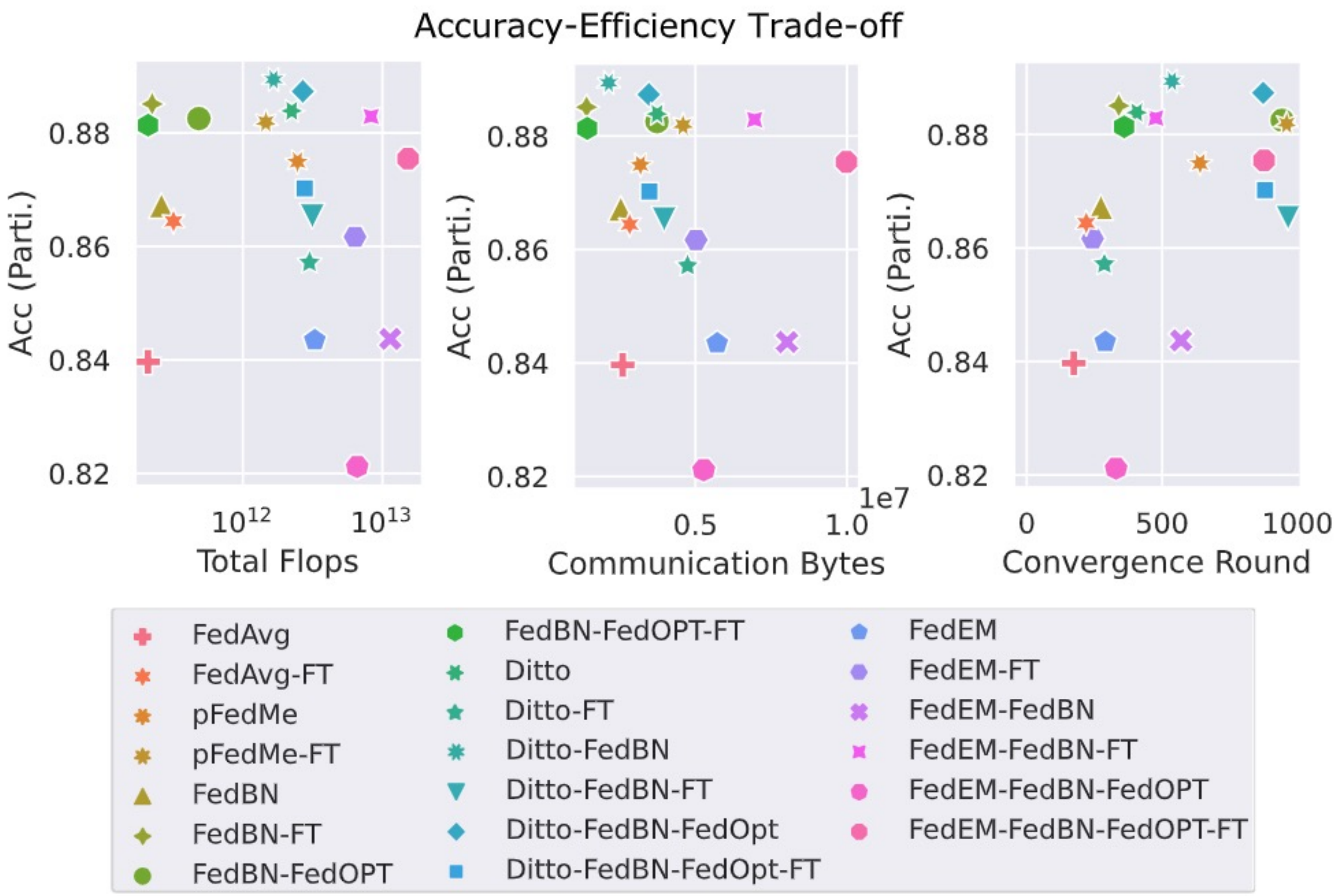
	FEMNIST, $s = 0.2$			SST-2			PUBMED		
	$\overline{Acc}'$	σ	$\overset{\smile}{Acc}$	$\overline{Acc}'$	σ	$\overset{\smile}{Acc}$	$\overline{Acc}'$	σ	$\overset{\smile}{Acc}$
Isolated	67.08	10.76	53.16	59.40	41.29	0.00	84.67	6.26	74.63
FedAvg	82.40	9.91	69.11	76.30	22.02	44.85	86.72	3.93	79.76
FedAvg-FT	85.17	8.69	72.34	75.36	27.67	31.08	86.71	3.86	80.57
pFedMe	86.50	8.52	75.00	65.08	26.59	27.75	86.35	4.43	78.76
pFedMe-FT	87.06	8.02	75.00	74.36	27.02	32.49	85.47	**3.06**	80.95
FedBN	85.38	8.19	74.26	76.30	22.02	44.85	**87.97**	3.42	81.77
FedBN-FT	87.65	**6.33**	**80.02**	68.50	26.83	29.17	87.02	3.47	80.13
FedBN-FedOPT	87.27	7.34	76.87	65.59	31.07	22.22	87.43	4.64	80.81
FedBN-FedOPT-FT	87.13	7.36	78.27	68.42	28.18	30.71	87.02	3.94	81.78
Ditto	87.18	7.52	78.23	49.94	40.81	0.00	86.85	3.98	80.44
Ditto-FT	84.30	8.16	73.95	54.34	39.26	0.00	87.10	3.52	80.46
Ditto-FedBN	**87.82**	7.19	77.78	49.44	41.80	0.00	87.75	3.70	81.82
Ditto-FedBN-FT	85.16	7.98	75.25	52.18	39.85	0.00	87.43	3.77	81.15
Ditto-FedBN-FedOpt	87.64	7.08	78.23	55.61	40.43	1.39	87.27	3.90	79.14
Ditto-FedBN-FedOpt-FT	85.71	7.91	75.81	53.16	34.75	9.72	87.10	3.79	80.93
FedEM	82.61	9.57	69.29	**76.53**	23.34	44.44	85.05	4.44	78.51
FedEM-FT	84.91	8.39	73.64	64.29	32.84	12.96	85.54	4.48	79.39
FedEM-FedBN	82.94	9.35	70.43	75.06	**18.48**	**53.33**	87.63	4.14	**82.54**
FedEM-FedBN-FT	87.09	9.24	76.36	64.33	35.72	8.59	85.68	4.33	79.44
FedEM-FedBN-FedOPT	80.48	11.02	64.84	72.66	27.18	34.17	87.11	4.24	80.32
FedEM-FedBN-FedOPT-FT	86.23	8.33	75.44	58.42	31.21	17.93	87.16	3.66	82.20

PFL Benchmark – System Efficiency

Metric

Total Flops
Communication Bytes
Convergence Round

- PFL pays large additional costs
- FT & FedOpt improve the convergence speeds



Reference of PFL

[1] Adaptive personalized federated learning. In arXiv 2020.

[2] Personalized federated learning with first order model optimization. In ICLR 2020.

[3] Parameterized Knowledge Transfer for Personalized Federated Learning. In NeurIPS 2021.

[4] Data-free knowledge distillation for heterogeneous federated learning. In ICML 2021.

[5] Ditto: Fair and robust federated learning through personalization. In ICML 2021.

[6] Personalized federated learning with moreau envelopes. In NeurIPS 2020.

[7] Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In NeurIPS 2020.

[8] FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. In ICLR 2021.

[9] Exploiting Shared Representations for Personalized Federated Learning. In ICML 2021

[10] An efficient framework for clustered federated learning. In NeurIPS 2020.

[11] Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. In TNNLS 2020

[12] Federated multi-task learning. In NeurIPS 2017.

[13] Federated multi-task learning under a mixture of distributions. In NeurIPS 2021.

[14] Tifl: A tier-based federated learning system. In ACM HPDC 2020.

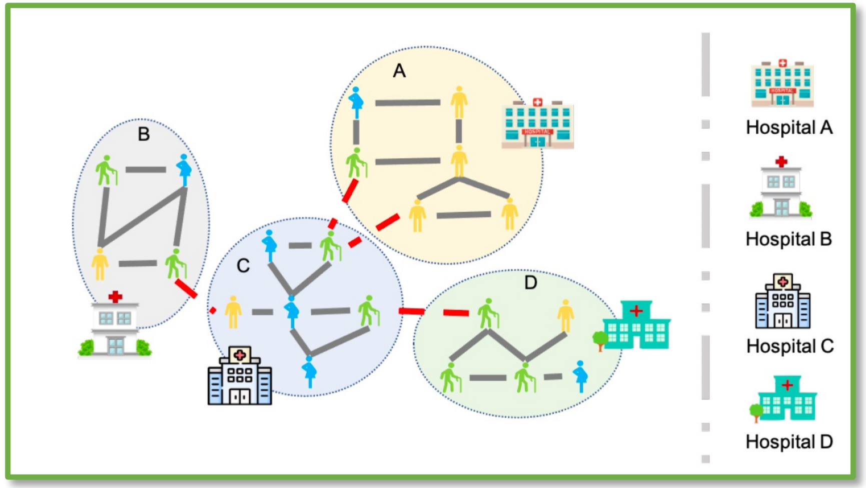
[15] pFL-Bench: A Comprehensive Benchmark for Personalized Federated Learning. In arXiv 2022

Agenda of Tutorial

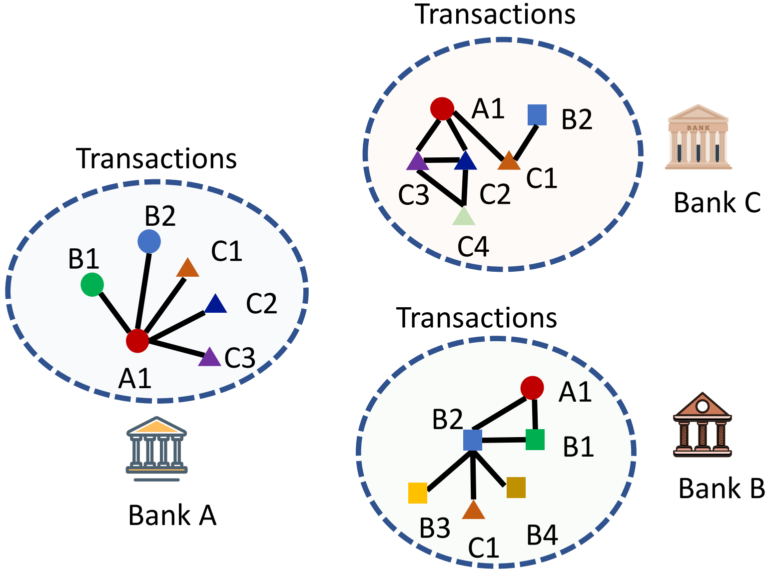
- Overview
- Personalized Federated Learning
- Federated Graph Learning
- Federated Hyperparameter Optimization
- Privacy Attacks

Federated Graph Learning (FGL)

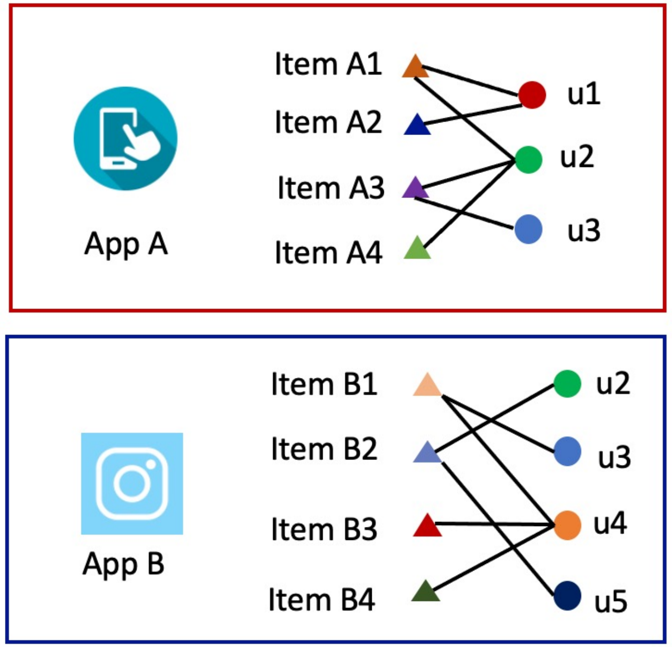
FGL Is Ubiquitous



Healthcare



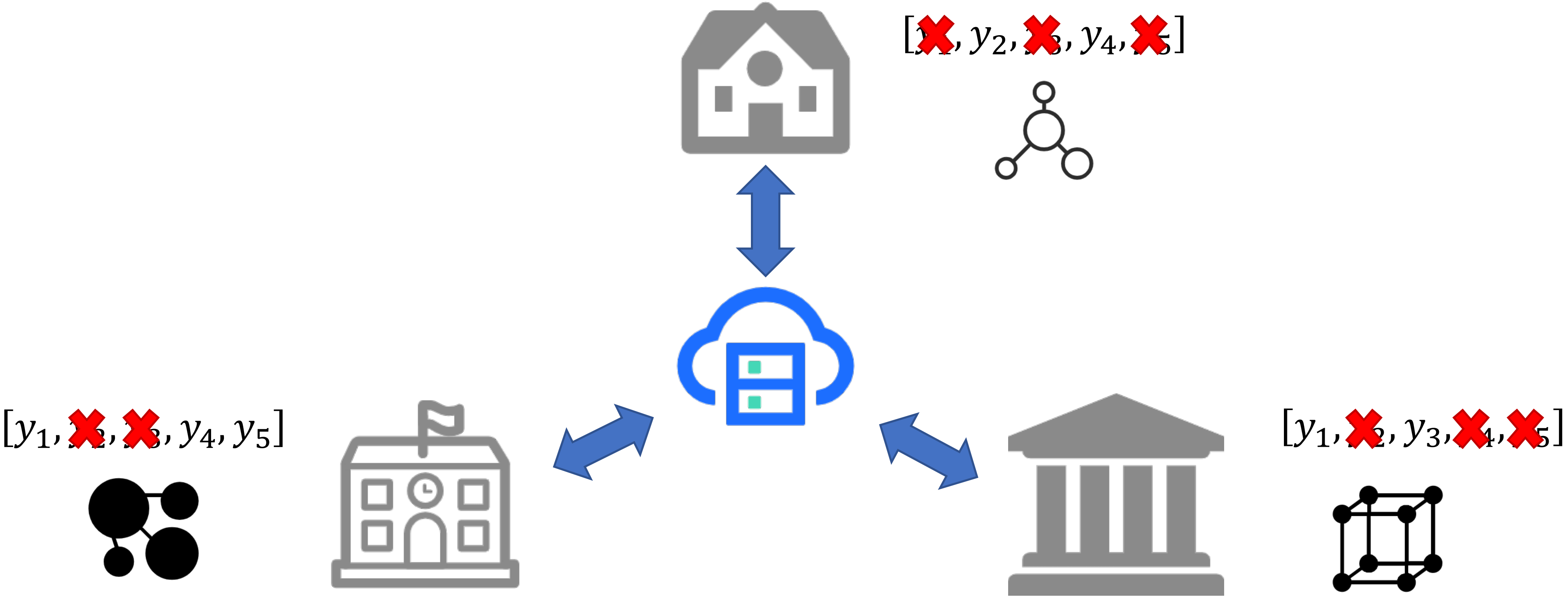
Anti-money laundering



Recommender system

Image source: Subgraph federated learning with missing neighbor generation. In NeurIPS 2021.

FGL Is Ubiquitous



Molecule-related research and drug discovery

FGL Scenario (1)

❑ Node classification

- Each client holds a subgraph
- Clients have no common node
- **Inter-subgraph edges are missing**

Healthcare

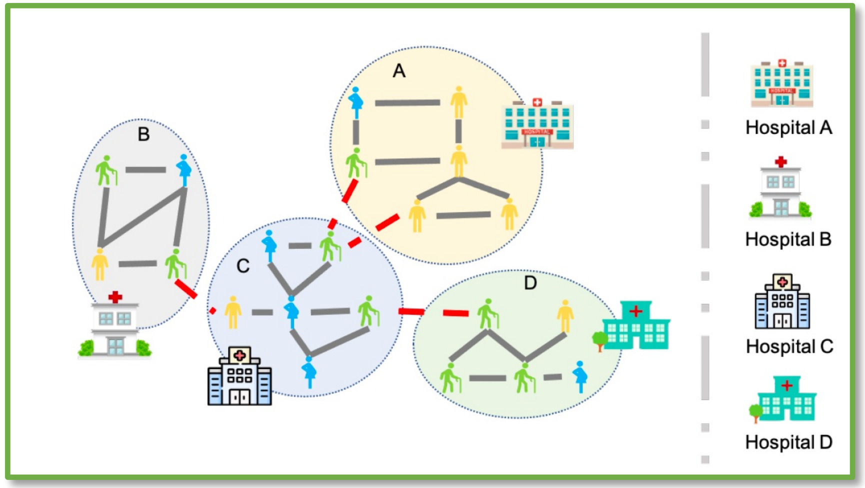


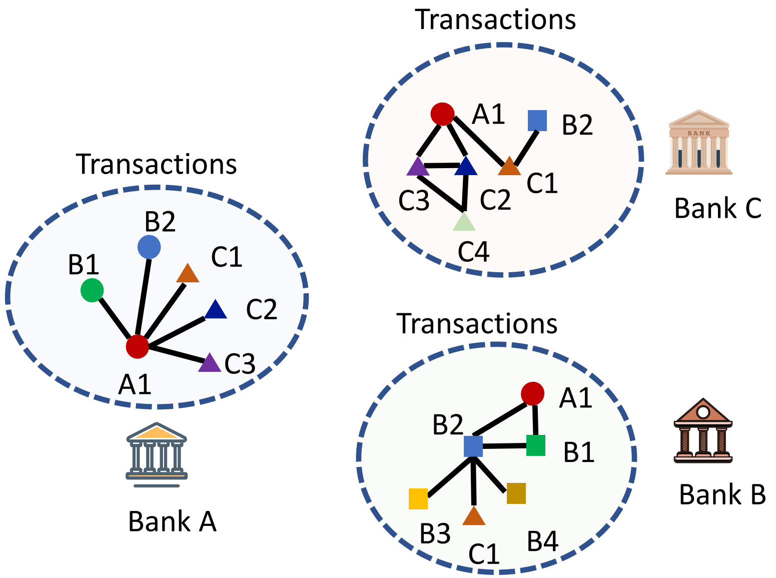
Image source: Subgraph federated learning with missing neighbor generation. In NeurIPS 2021.

FGL Scenario (2)

❑ Node classification

- Each client holds a subgraph
- Clients have common nodes
- **Intra-subgraph edges may be missing**

Anti-money laundering

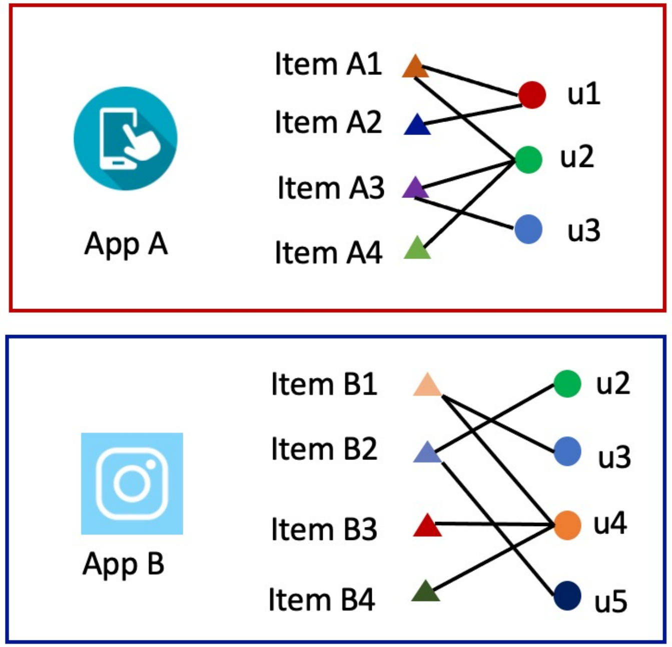


FGL Scenario (3)

❑ Link prediction

- Each client holds a bipartite graph
- Clients have common node
- **Inter-subgraph edges are missing**

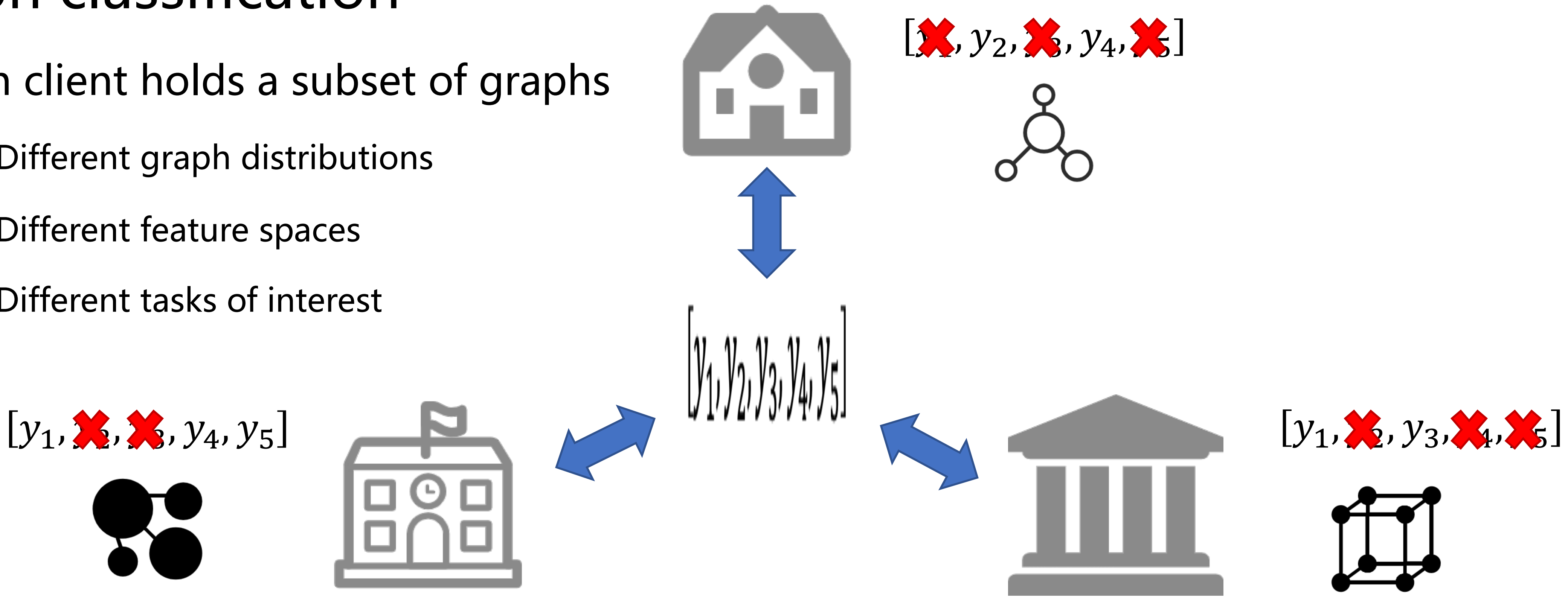
Recommender system



FGL Scenario (4)

❑ Graph classification

- Each client holds a subset of graphs
  - Different graph distributions
  - Different feature spaces
  - Different tasks of interest

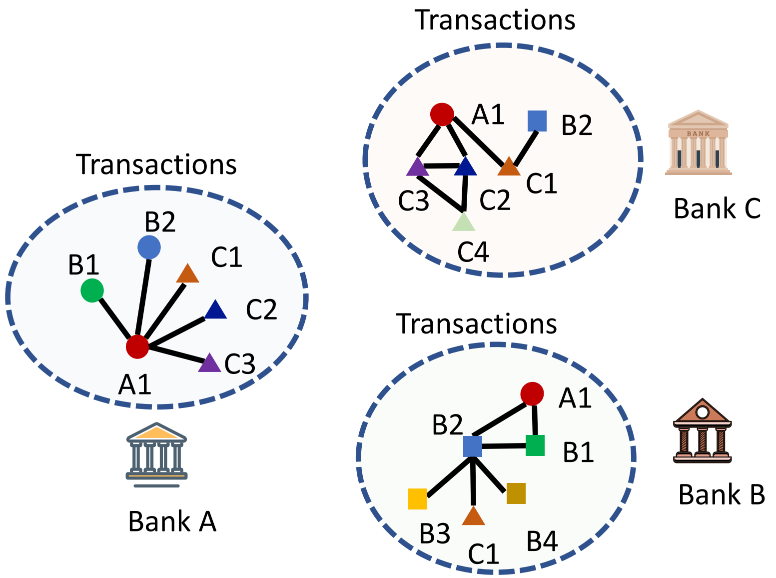


Molecule-related research and drug discovery

Challenges in FGL

❑ Non-IIDness

- The same node in different clients may have different neighbors

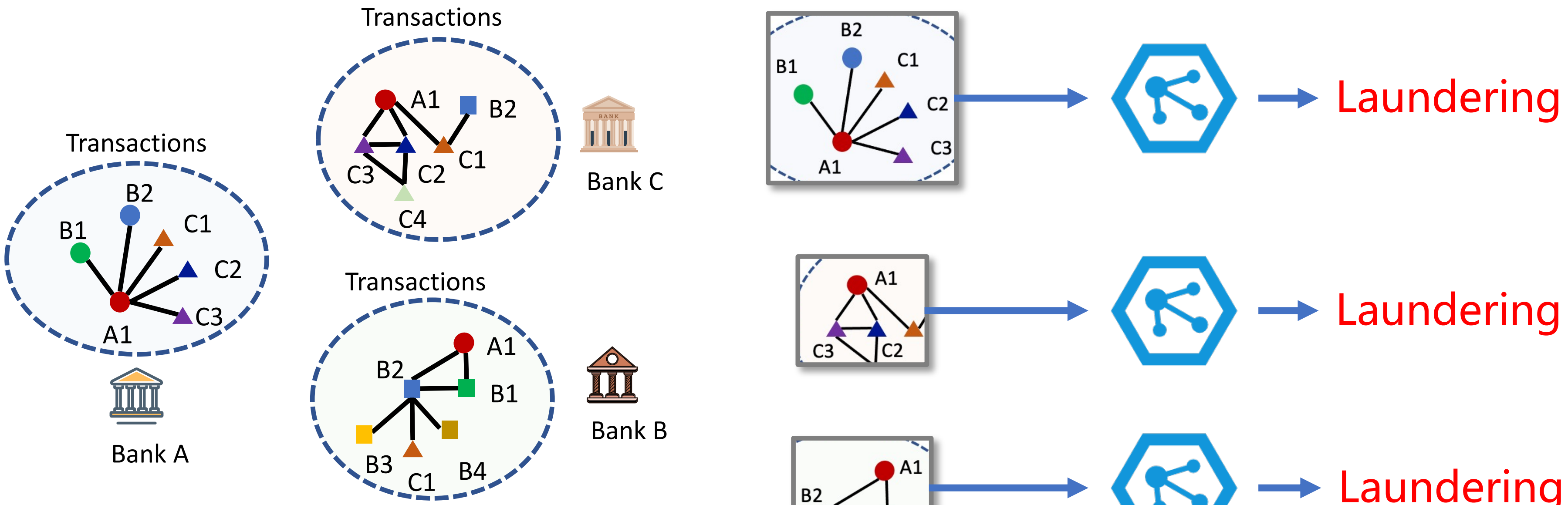


Anti-money laundering

Challenges in FGL

❑ Non-IIDness

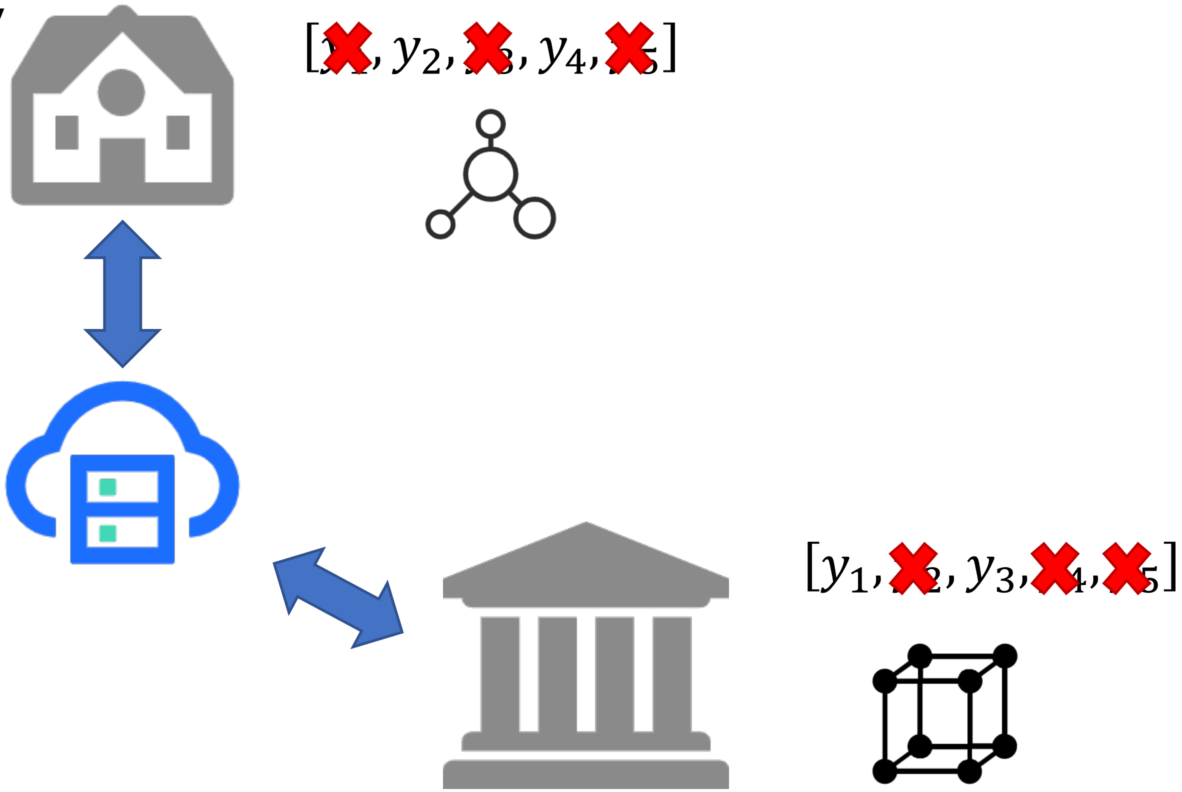
- The same node in different clients may have different neighbors
- Models have to associate different patterns with the same label

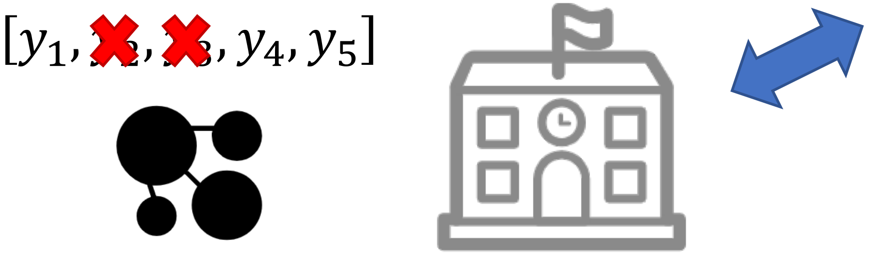


Anti-money laundering

Challenges in FGL

- More general heterogeneity
  - Different feature spaces
  - Different tasks of interest

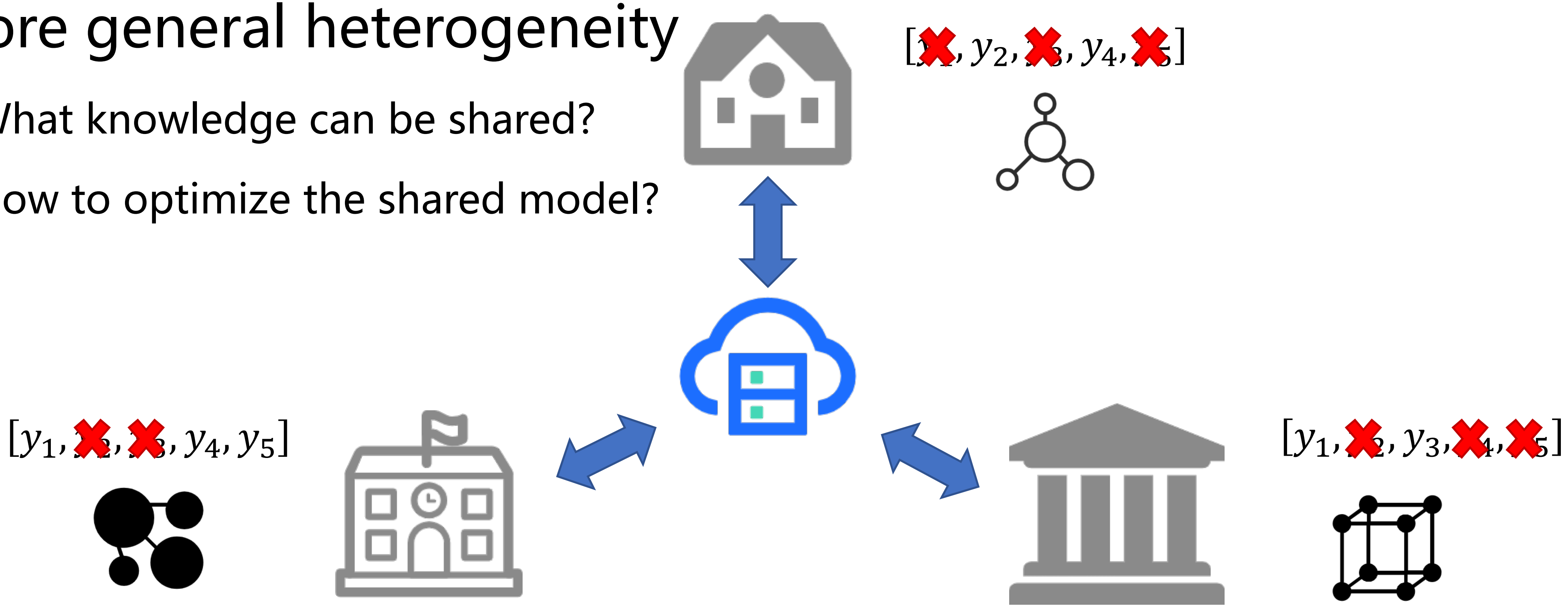




Molecule-related research and drug discovery

Challenges in FGL

- More general heterogeneity
  - What knowledge can be shared?
  - How to optimize the shared model?



Molecule-related research and drug discovery

Study FGL with FederatedScope-GNN (FS-G) [1]

- ❑ **Datasets**
  - **Very comprehensive in terms of tasks and types of heterogeneity**
- ❑ GNN models
  - Out-of-the-box GNNs and Client-specific neural architecture
- ❑ FGL Algorithms
  - FL participants exchange heterogeneous information and have rich behaviors

Creating FGL Datasets

❑ Community-based splitters

- Apply community detection algorithms to partition a graph into several clusters
  - E.g., Louvain [2] and METIS [3]
- Assign clusters to clients, optionally balancing their #node
  - Nodes in the same client are densely connected

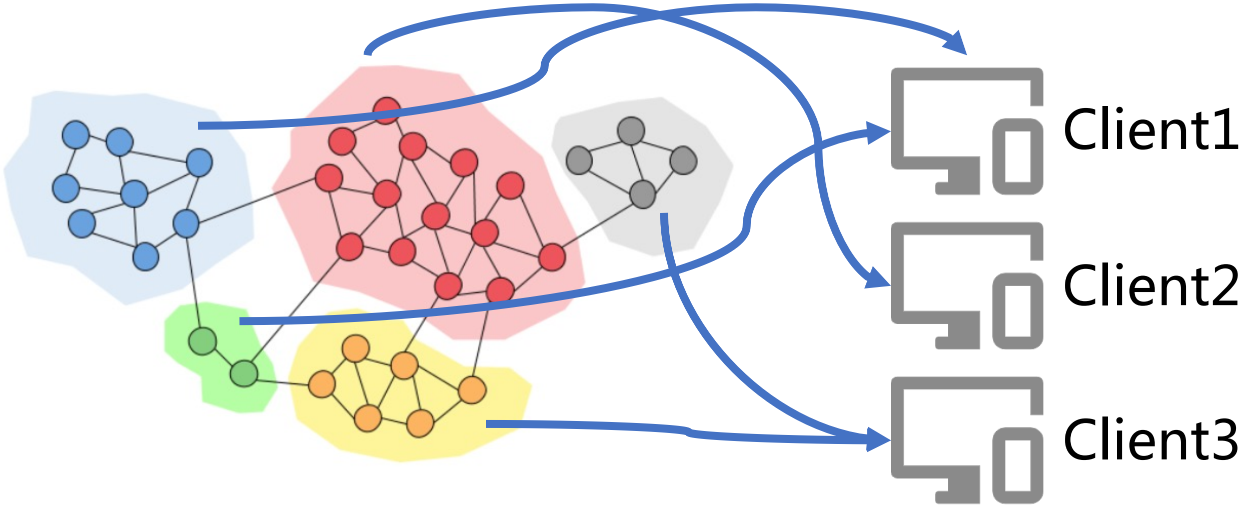


Image source: Community Detection Algorithms. In TDS.

Creating FGL Datasets

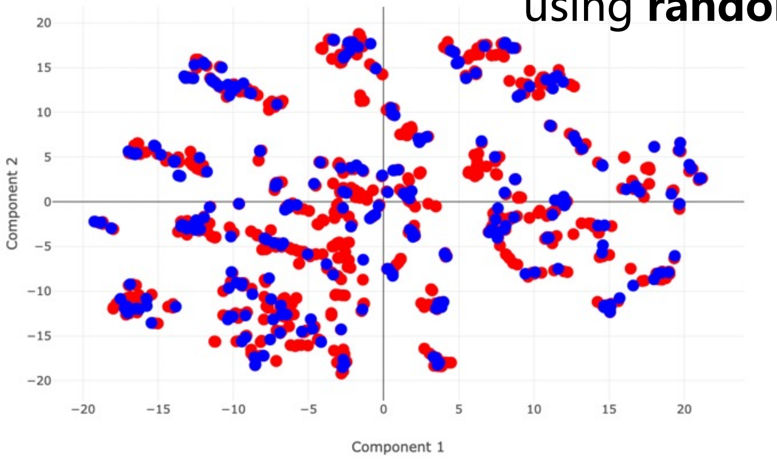
- ❑ Randomness-based splitters
  - The node set of original graph is randomly split into N subsets
  - Subgraph of each client is deduced from its nodes
- ❑ Metadata-based splitters
  - E.g., split a citation network by venue
  - E.g., split a user-item interaction network by user

Creating FGL Datasets

❑ Instance space-based splitters

- E.g., sort all molecular graphs by their scaffold, and then each client is assigned with a segment of the sorted list
- Useful for creating covariate shift

PCA chemical space of two sets using **random v.s. scaffold** split



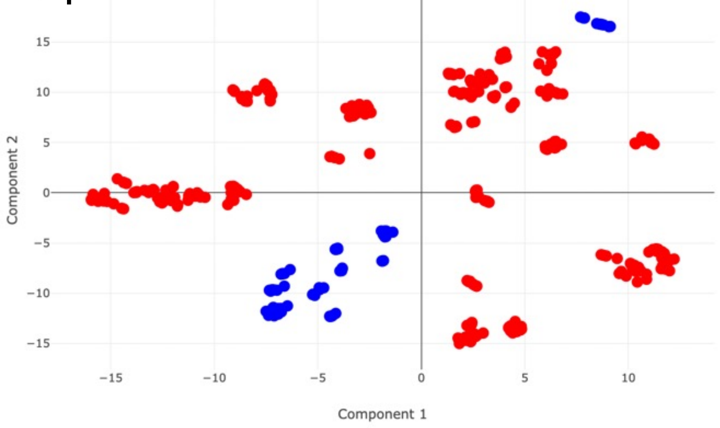


Image source: Introduction to scaffold splitting. In Oloren Blog Home.

Creating FGL Datasets

❑ Label space-based splitters

- E.g., in relation prediction task defined on a knowledge graph, triplets are split into clients by latent dirichlet allocation (LDA) [4]
- This creates label distribution skew

❑ Task-based splitters

- E.g., one client has molecules labeled with their toxicity, another client has molecules labeled with their excitation energy.
- This creates federated hetero-task learning [5, 6]

Creating FGL Datasets

Task	Domain	Dataset	Splitter	# Graph	Avg. # Nodes	Avg. # Edges	# Class	Evaluation
Node-level	Citation network	Cora	*random&community*	1	2,708	5,429	7	ACC
	Citation network	CiteSeer	*random&community*	1	4,230	5,358	6	ACC
	Citation network	PubMed	*random&community*	1	19,717	44,338	5	ACC
	Citation network	FedDBLP	*meta*	1	52,202	271,054	4	ACC
Link-level	Recommendation System	Ciao	*meta*	28	5,875.68	20,189.29	6	ACC
	Recommendation System	Taobao	*meta*	3	443,365	2,015,558	2	ACC
	Knowledge Graph	WN18	*label_space*	1	40,943	151,442	18	Hits@n
	Knowledge Graph	FB15k-237	*label_space*	1	14,541	310,116	237	Hits@n
Graph-level	Molecule	HIV	*instance_space*	41,127	25.51	54.93	2	ROC-AUC
	Proteins	Proteins	*instance_space*	1,113	39.05	145.63	2	ACC
	Social network	IMDB	*label_space*	1,000	19.77	193.06	2	ACC
	Multi-task	Mol	*multi_task*	18,661	55.62	1,466.83	-	ACC

FS-G [1] provides off-the-shelf FGL datasets.

Creating FGL Datasets

Task	Domain	Dataset	Splitter	# Graph	Avg. # Nodes	Avg. # Edges	# Class	Evaluation
Node-level	Citation network	Cora	*random&community*	1	2,708	5,429	7	ACC
	Citation network	CiteSeer	*random&community*	1	4,230	5,358	6	ACC
	Citation network	PubMed	*random&community*	1	19,717	44,338	5	ACC
	Citation network	FedDBLP	*meta*	1	52,202	271,054	4	ACC
Link-level	Recommendation System	Ciao	*meta*	28	5,875.68	20,189.29	6	ACC
	Recommendation System	Taobao	*meta*	3	443,365	2,015,558	2	ACC
	Knowledge Graph	WN18	*label_space*	1	40,943	151,442	18	Hits@n
	Knowledge Graph	FB15k-237	*label_space*	1	14,541	310,116	237	Hits@n
Graph-level	Molecule	HIV	*instance_space*	41,127	25.51	54.93	2	ROC-AUC
	Proteins	Proteins	*instance_space*	1,113	39.05	145.63	2	ACC
	Social network	IMDB	*label_space*	1,000	19.77	193.06	2	ACC
	Multi-task	Mol	*multi_task*	18,661	55.62	1,466.83	-	ACC

```yaml
federate:
  client_num: 5
data:
  root: data/
  type: cora # which standalone graph dataset to use
  splitter: louvain # community detection-based
  splitter_args: [{'delta': 20}]
  batch_size: 1 # Full batch training
```

Example: construct a FGL node classification task with 5 clients from citation network Cora.

Study FGL with FederatedScope-GNN (FS-G) [1]

- Datasets
  - Very comprehensive in terms of tasks and types of heterogeneity
- **GNN models**
  - **Out-of-the-box GNNs and Client-specific neural architecture**
- FGL Algorithms
  - FL participants exchange heterogeneous information and have rich behaviors

Graph Neural Networks (GNN)

❑ Given a graph $G(V, E)$

- V is the node set, and $E = \{(i,j)|i,j \in V\}$ is the edge set
- Each node is associated with a k-dimensional feature vector; all node features are denoted by $X \in \mathfrak{R}^{|V|\times k}$
- Adjacency A has $A_{ij} = 1$ if $(i,j) \in E$, otherwise $A_{ij} = 0$

❑ What GNN does

- Generate node embeddings
- Encode both node features and graph structure

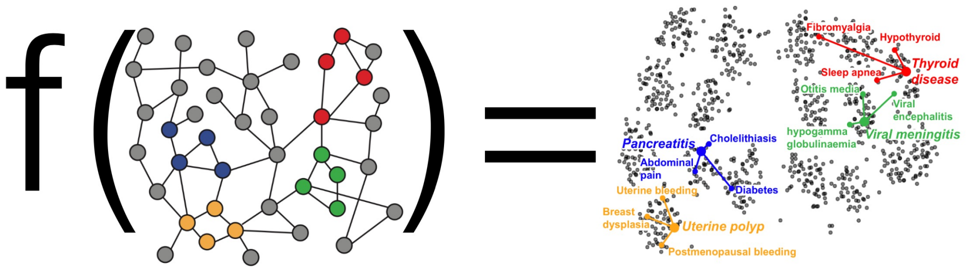
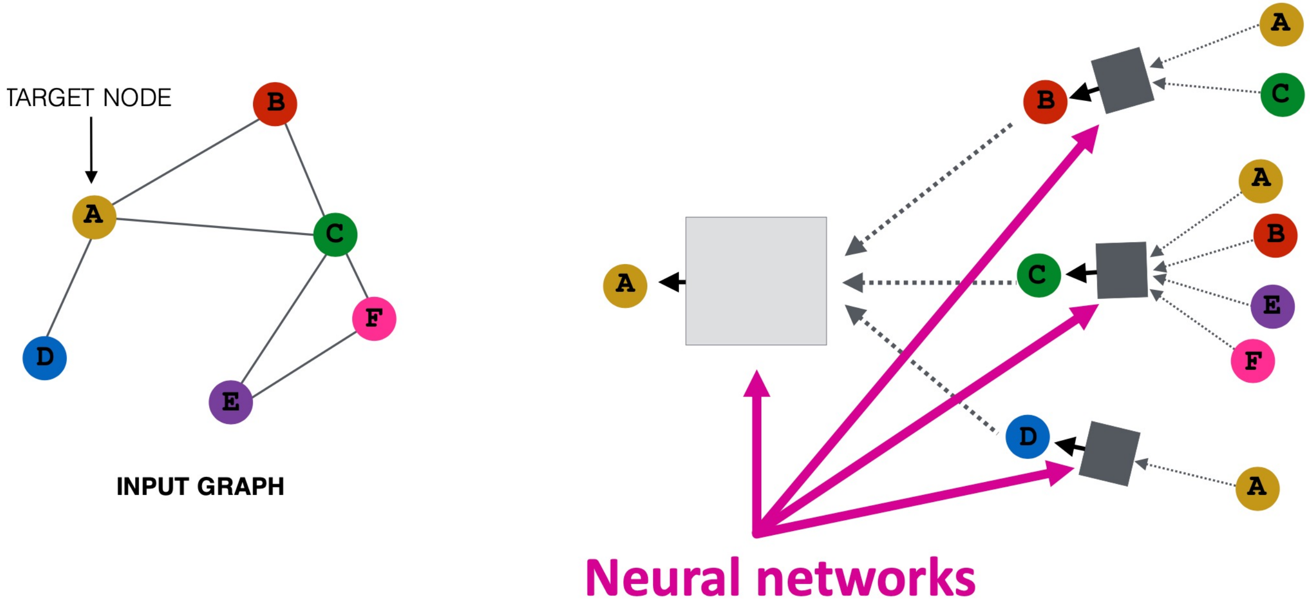


Image source: Machine Learning with Graphs. In Stanford CS224W.

Graph Neural Networks (GNN)

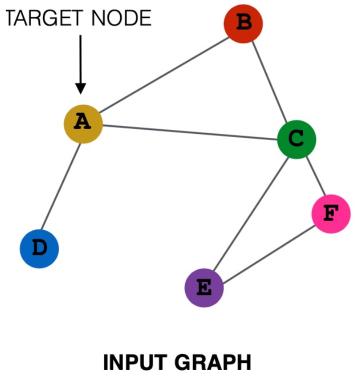
❑ Key idea behind GNN

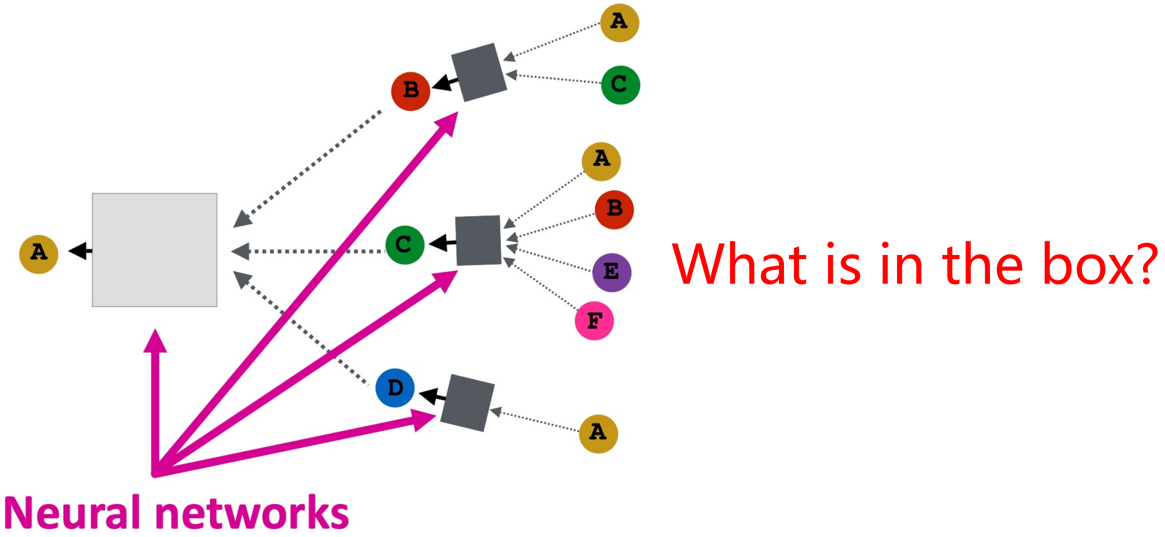
- Generate node embeddings by aggregating information from neighborhood
- Use neural networks to parameterize the aggregation procedure



[Image source](): Machine Learning with Graphs. In Stanford CS224W.

Graph Neural Networks (GNN)





- General form $h_i^{(l+1)} = \gamma^{(l)}\left(h_i^{(l)}, \text{agg}_{j\in N(i)}\phi^{(l)}\left(h_i^{(l)}, h_j^{(l)}\right)\right)$
  - Let $h_i^{(l)}$ denote the embedding of node i at the l-th layer, where $h_i^{(0)} = X_i$ is defined as the raw node feature
  - $\gamma^{(l)}(\quad)$ and $\phi^{(l)}$ are neural networks (e.g., MLP) for feature mapping
  - "agg" is an aggregation operator, e.g., elementwise mean/min/max

ModelZoo of FederatedScope-GNN

❑ Out-of-the-box GNNs

- Implemented based on PyG
- GCN [7], GIN [8], GAT [9], GraphSage [10], GPRGNN [11], etc.
- Various readout choices, e.g., elementwise min/mean/max

```
model:
  type: sage # to use GraphSage
  hidden: 64 # dimension of node embeddings
  dropout: 0.5
  out_channels: 7
  task: NodeClassification
```

Example: Apply GraphSage
to node-level task.

```
model:
  type: gcn # to use GCN
  hidden: 64
  out_channels: 2
  task: GraphClassification
  graph_pooling: mean # readout operator
```

Example: Apply GCN
to graph-level tasks.

ModelZoo is Extendable

❑ Contribute and use novel GNN model

```
from federatedscope.register import register_model

class MyNet(torch.nn.Module):
    def __init__(self,
                 in_channels,
                 out_channels):
        super(MyNet, self).__init__()
        self.convs = ModuleList()
        for i in range(2):
            if i == 0:
                self.convs.append(GCNConv(in_channels, 64))
            elif (i + 1) == 2:
                self.convs.append(GCNConv(64, out_channels))
            else:
                self.convs.append(GCNConv(64, 64))

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        for i, conv in enumerate(self.convs):
            x = conv(x, edge_index)
            if (i + 1) == len(self.convs):
                break
            x = F.relu(x)
        return x

def load_my_net(model_config, data):
    model = MyNet(data.x.shape[-1],
                  model_config.out_channels)
    return model

def call_my_net(model_config, data):
    if model_config.type == "mynet":
        model = load_my_net(model_config, data)
        return model

register_model("mynet", call_my_net)
```

Implement your model based on PyG and put the .py in contrib/model/example.py

Register this model in FS-G

ModelZoo is Extendable

- ❑ Contribute and use novel GNN model

```python
from federatedscope.register import register_model

class MyNet(torch.nn.Module):
    def __init__(self,
                 in_channels,
                 out_channels):
        super(MyNet, self).__init__()
        self.convs = ModuleList()
        for i in range(2):
            if i == 0:
                self.convs.append(GCNConv(in_channels, 64))
            elif (i + 1) == 2:
                self.convs.append(GCNConv(64, out_channels))
            else:
                self.convs.append(GCNConv(64, 64))

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        for i, conv in enumerate(self.convs):
            x = conv(x, edge_index)
            if (i + 1) == len(self.convs):
                break
            x = F.relu(x)
        return x

def load_my_net(model_config, data):
    model = MyNet(data.x.shape[-1],
                  model_config.out_channels)
    return model

def call_my_net(model_config, data):
    if model_config.type == "mynet":
        model = load_my_net(model_config, data)
        return model

register_model("mynet", call_my_net)
```

Register this model in FS-G

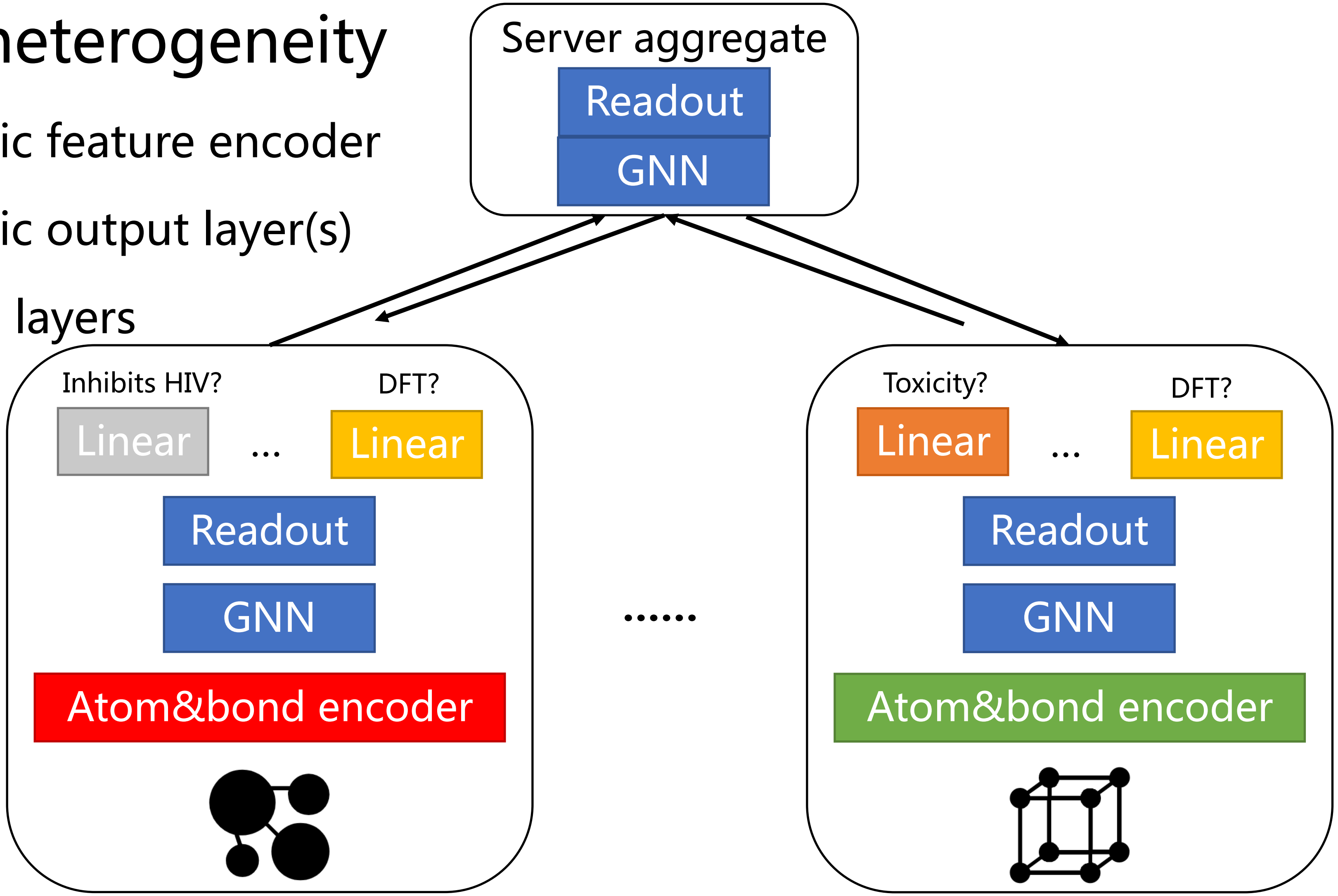
Implement your model based on PyG and put the .py in contrib/model/example.py

```yaml
model:
  type: mynet
  out_channel: 2
  task: NodeClassification
```

Then we could use it by specifying in the .yaml

Client-specific Neural Architecture

- ❑ Handling heterogeneity
  - Client-specific feature encoder
  - Client-specific output layer(s)
  - Shared GNN layers



Client-specific Neural Architecture

❑ Handling heterogeneity

- Client-specific feature encoder
- Client-specific output layer(s)
- Shared GNN layers

```yaml
# declare which model parameters should not be aggregated
personalization:
  local_param: ['encoder_atom', 'encoder', 'clf']
```

Then only the parameters of GNN layers would be exchanged and aggregated

```yaml
# per_client.yaml
client_1:
  model:
    out_channels: 2
    task: graphClassification
  criterion:
    type: CrossEntropyLoss
  train:
    optimizer:
      lr: 0.1
  eval:
    metrics: ['acc']
client_2:
  model:
    out_channels: 1
    task: graphRegression
  criterion:
    type: MSELoss
  train:
    optimizer:
      lr: 0.01
  eval:
    metrics: ['mse']
...
```

Users can further specify client-wise configurations

Capability of Handling Hetero-task

❑ Empirical study on Graph-DT [6]

- 16 clients, each has a graph dataset picked from TUDataset/MoleculeNet
- All are molecular graphs, but each dataset has specific atom attributes and tasks
- Clients share the MPNN layers but have client-specific atom encoders and output layers

	Overall (%)
FedAvg	1.69%
FedAvg+FT	2.13%
FedProx	−3.99%
FedBN	5.93%
FedBN+FT	8.48%
Ditto	−5.41%
FedMAML	4.64%

Average improvement ratio
w.r.t. "isolated training" baseline

Study FGL with FederatedScope-GNN (FS-G) [1]

❑ Datasets

- Very comprehensive in terms of tasks and types of heterogeneity

❑ GNN models

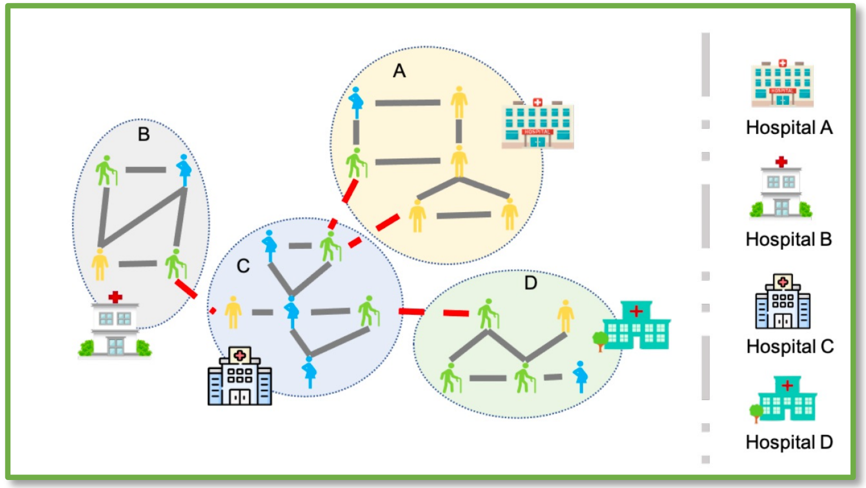
- Out-of-the-box GNNs and Client-specific neural architecture

❑ **FGL Algorithms**

- **FL participants exchange heterogeneous information and have rich behaviors**

AlgoZoo of FederatedScope-GNN

- A representative FGL algorithm: FedSage+ [12]
  - Attempts to mend the graph by generating missing neighbor(s)
  - Mainly focuses on this scenario:



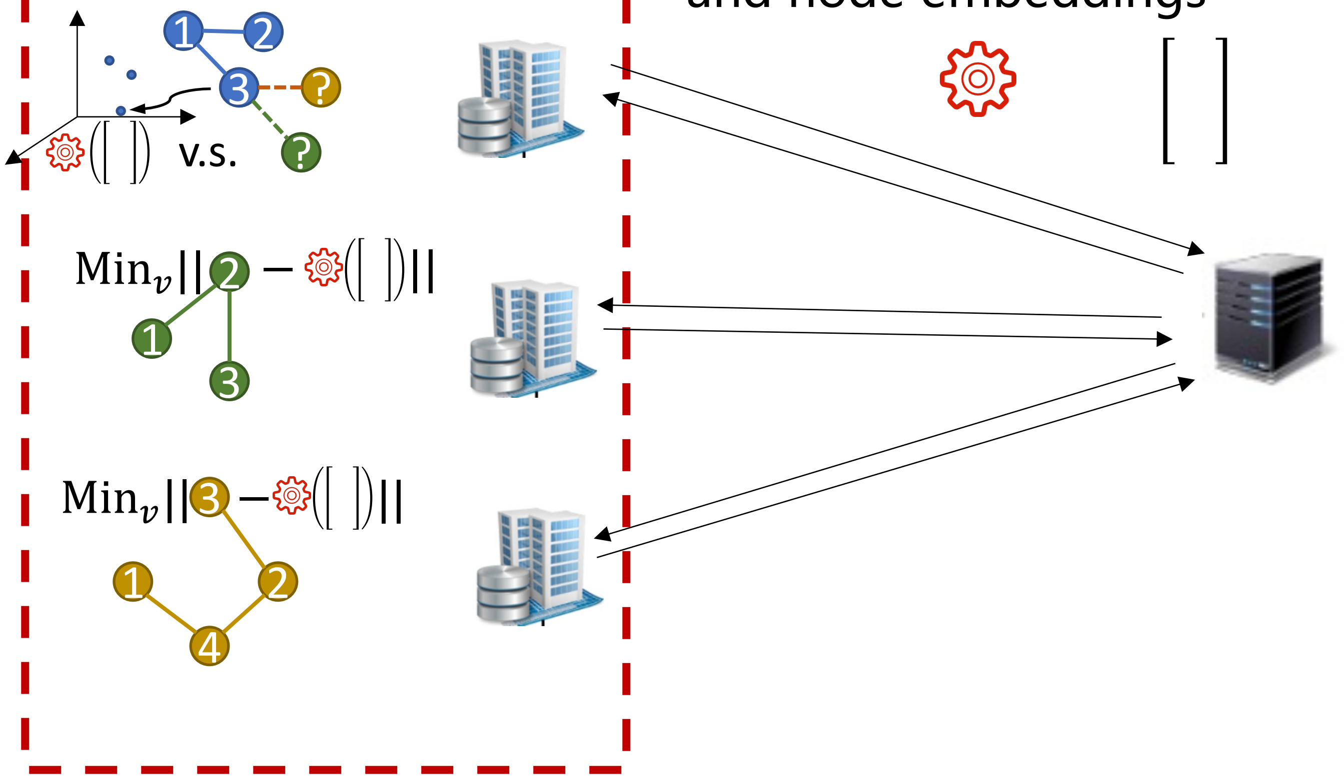
- Each client holds a subgraph
- Clients have no common node
- **Inter-subgraph edges are missing**

Image source: Subgraph federated learning with missing neighbor generation. In NeurIPS 2021.

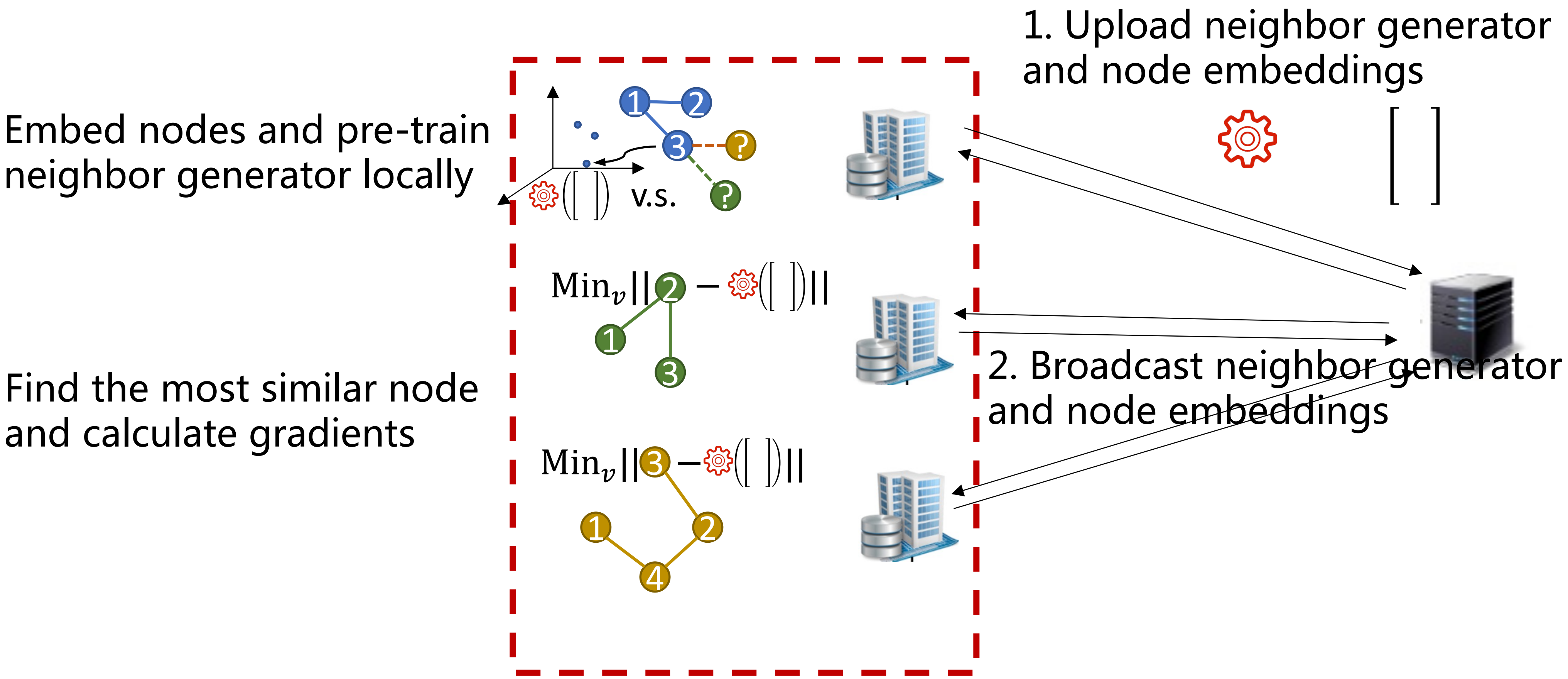
Heterogeneous Information and Rich Behaviors

Embed nodes and pre-train neighbor generator locally

1. Upload neighbor generator and node embeddings



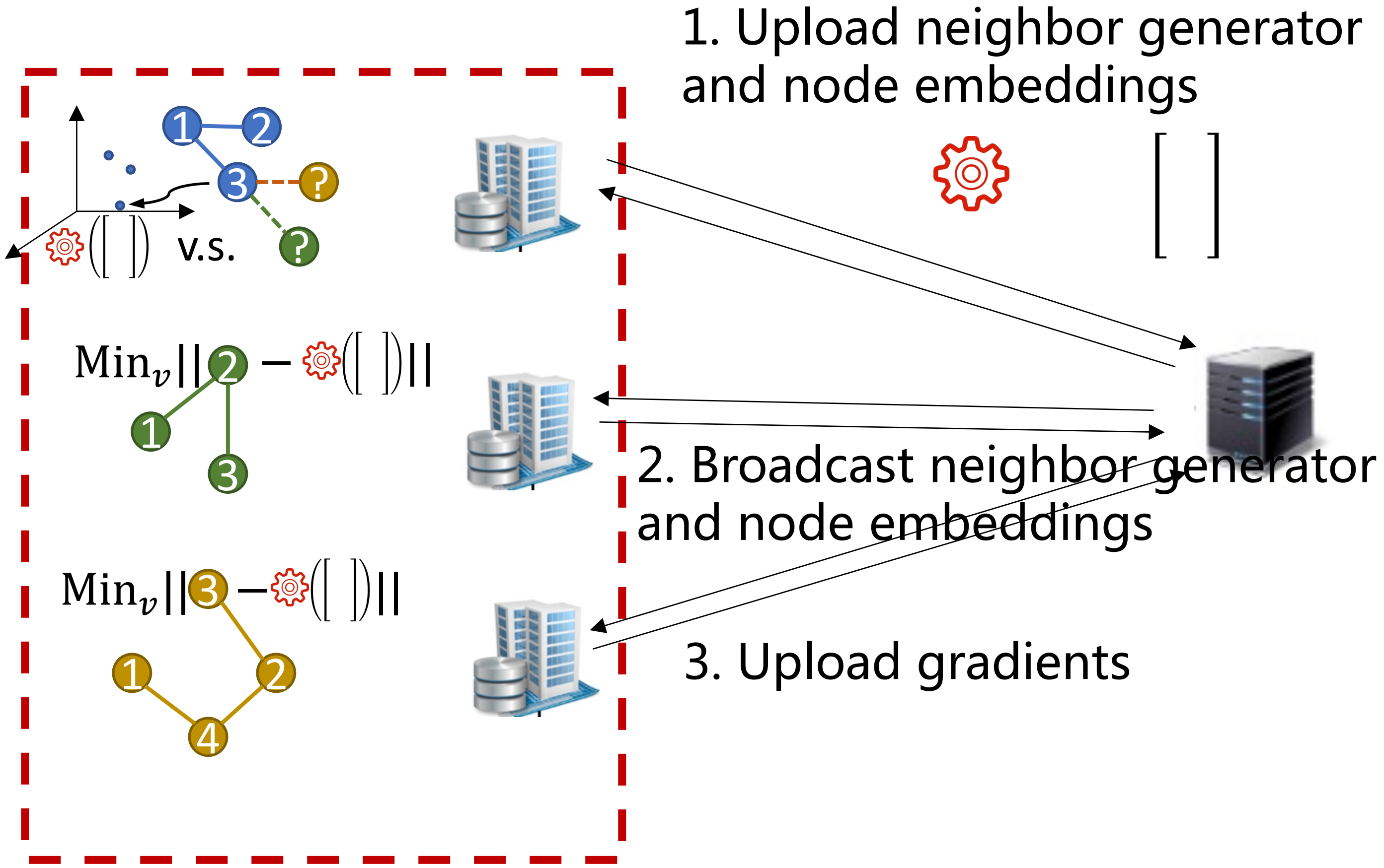
Heterogeneous Information and Rich Behaviors



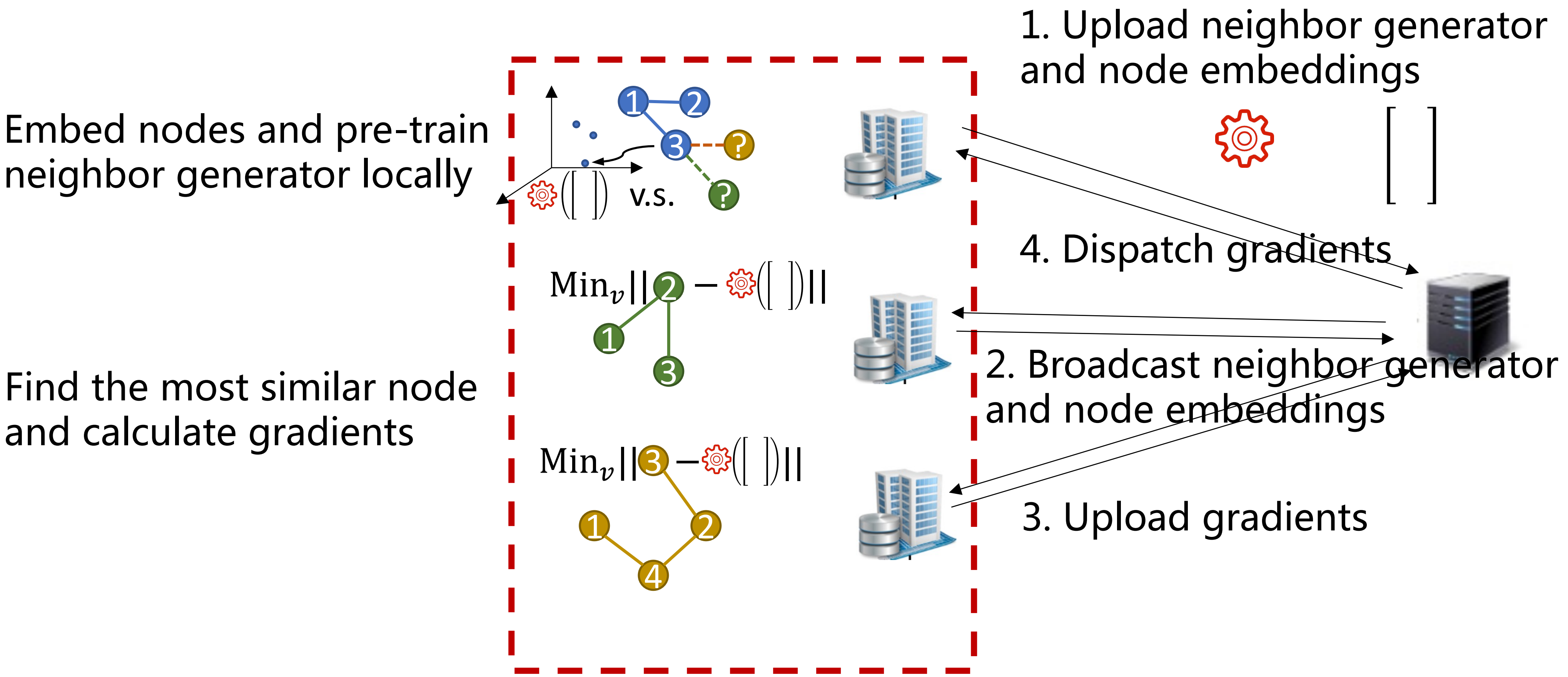
Heterogeneous Information and Rich Behaviors

Embed nodes and pre-train neighbor generator locally

Find the most similar node and calculate gradients



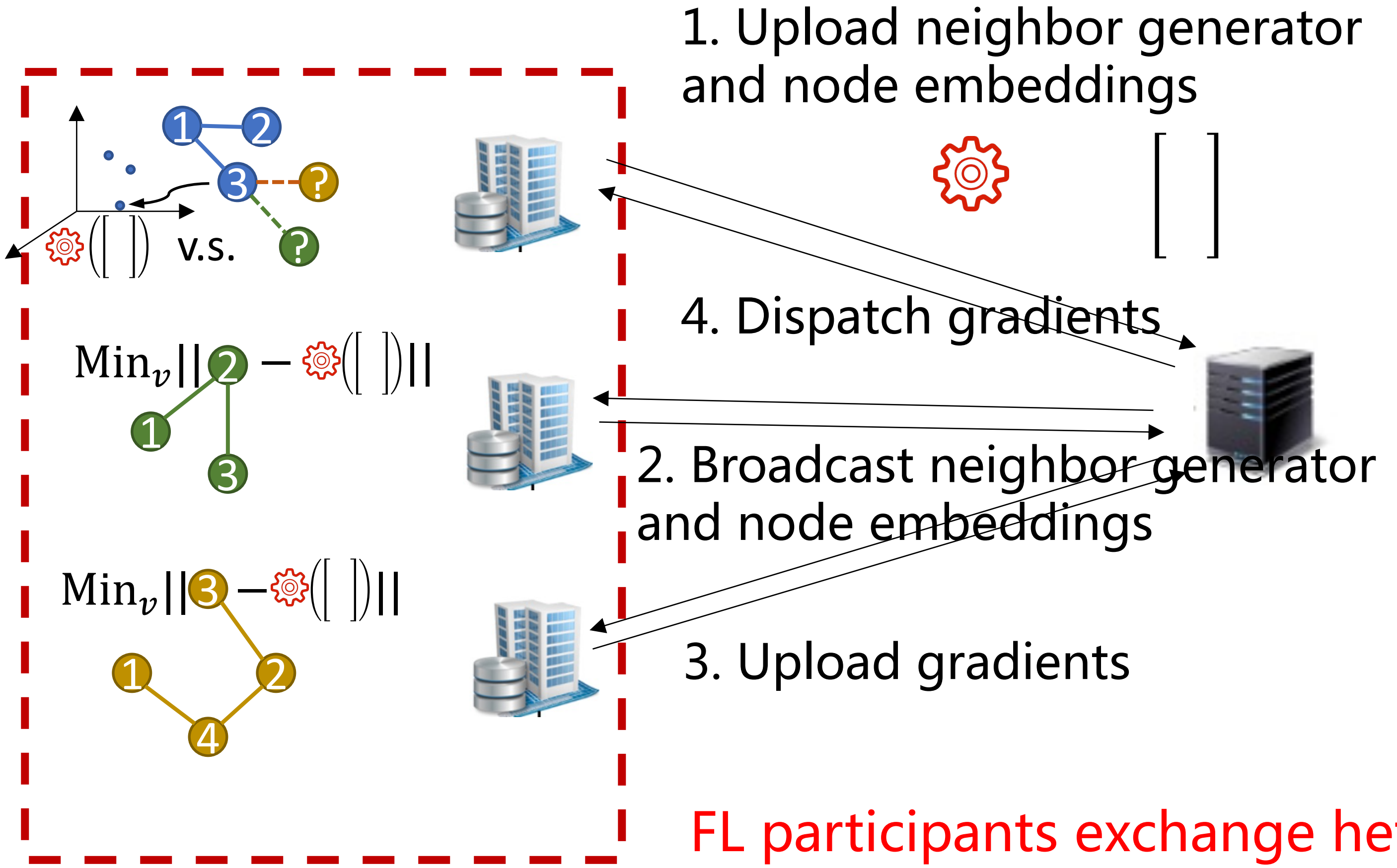
Heterogeneous Information and Rich Behaviors



Heterogeneous Information and Rich Behaviors

Embed nodes and pre-train neighbor generator locally

Find the most similar node and calculate gradients

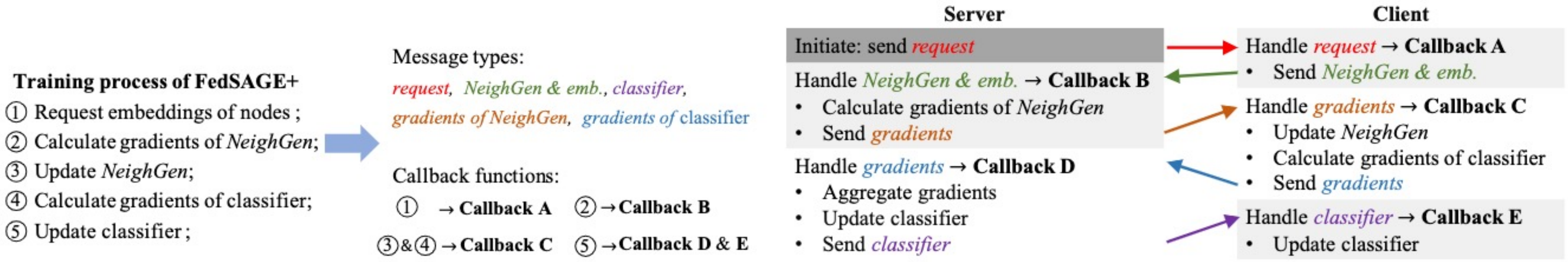


FL participants exchange heterogeneous information and have rich behaviors!

Implementing FGL Algorithms in FS-G

❑ Event-driven framework

- Define various messages regarding the exchanged information
- Frame the algorithmic procedure into multiple event handlers



How we implement FedSage+ in the AlgoZoo of FS-G.

Empirical Study Using AlgoZoo of FS-G

❑ Compare FedSage+ with GraphSage

- Identical neural architecture, but
- GraphSage is federally learned by FedAvg without graph mending

	Cora		CiteSeer		PubMed	
	random	community	random	community	random	community
GraphSAGE	85.42±1.80	87.19±1.28	76.86±1.38	77.80±1.03	86.45±0.43	86.87±0.53
FedSage+	85.07±1.20	87.68±1.55	78.04±0.91	77.98±1.23	88.19±0.32	87.94±0.27

Mean test accuracy $\pm$ standard deviation

Future Directions

❑ Subgraph completion

- How to borrow information from other subgraphs in a privacy-preserving manner?

❑ Self-supervised learning in the FL setting

- Non-IIDness issue is exacerbated in existing observations

❑ Federated molecular property prediction

- Both molecules and their labels can be private

References of FGL

[1] FederatedScope-GNN: Towards a Unified, Comprehensive and Efficient Package for Federated Graph Learning. In KDD 2022.
[2] Fast unfolding of communities in large networks. In Journal of Statistical Mechanics: Theory and Experiment 2008.
[3] Multilevel k-way hypergraph partitioning. In VLSI design 2000.
[4] Latent dirichlet allocation. In JMLR 2003.
[5] Federated graph classification over non-iid graphs. In NeurIPS 2021.
[6] A Benchmark for Federated Hetero-Task Learning. In arXiv 2022.
[7] Semi-Supervised Classification with Graph Convolutional Networks. In ICLR 2017.
[8] How Powerful are Graph Neural Networks?. In ICLR 2018.
[9] Graph Attention Networks. In ICLR 2018.
[10] Inductive representation learning on large graphs. In NeurIPS 2017.
[11] Adaptive Universal Generalized PageRank Graph Neural Network. In ICLR 2020.
[12] Subgraph federated learning with missing neighbor generation. In NeurIPS 2021.

Agenda of Tutorial

- Overview
- Personalized Federated Learning
- Federated Graph Learning
- Federated Hyperparameter Optimization
- Privacy Attacks

Federated Hyperparameter Optimization (FedHPO)

Hyperparameter Optimization (HPO)

- ❑ Problem definition

$$\min_{\lambda \in \Lambda_1 \times \cdots \times \Lambda_K} f(\lambda)$$

- ❑ Black-box function f
  - Domain
    - i.e., search space
    - e.g., learning rate $\in \Lambda_1 = [0.001, 0.1]$
    - e.g., batch size $\in \Lambda_2 = \{16, 32, 64\}$
  - Function evaluation
    - Executing the corresponding algorithm with the given hyperparameter configuration λ
    - Producing the output, e.g., validation loss

Hyperparameter Optimization (HPO)

❑ Problem definition

$$\min_{\lambda \in \Lambda_1 \times \cdots \times \Lambda_K} f(\lambda)$$

❑ Black-box function f

- Domain
  - i.e., search space
  - e.g., learning rate $\in \Lambda_1 = [0.001, 0.1]$
  - e.g., batch size $\in \Lambda_2 = \{16, 32, 64\}$

Continuous, ordinal, categorical, etc.

- Function evaluation
  - Executing the corresponding algorithm with the given hyperparameter configuration λ
  - Producing the output, e.g., validation loss

Non-analytic, non-convex, non-smooth, time-consuming

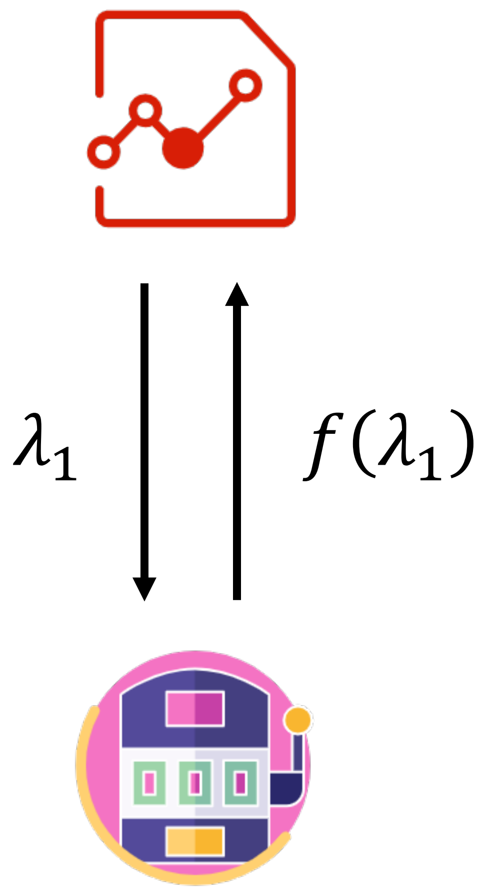
HPO: A Black-box Optimization View

❑ Problem definition

$$\min_{\lambda \in \Lambda_1 \times \cdots \times \Lambda_K} f(\lambda)$$

❑ Black-box function f

- Domain
  - i.e., search space
  - e.g., learning rate $\in \Lambda_1 = [0.001, 0.1]$
  - e.g., batch size $\in \Lambda_2 = \{16, 32, 64\}$
- Function evaluation
  - Executing the corresponding algorithm with the given hyperparameter configuration λ
  - Producing the output, e.g., validation loss



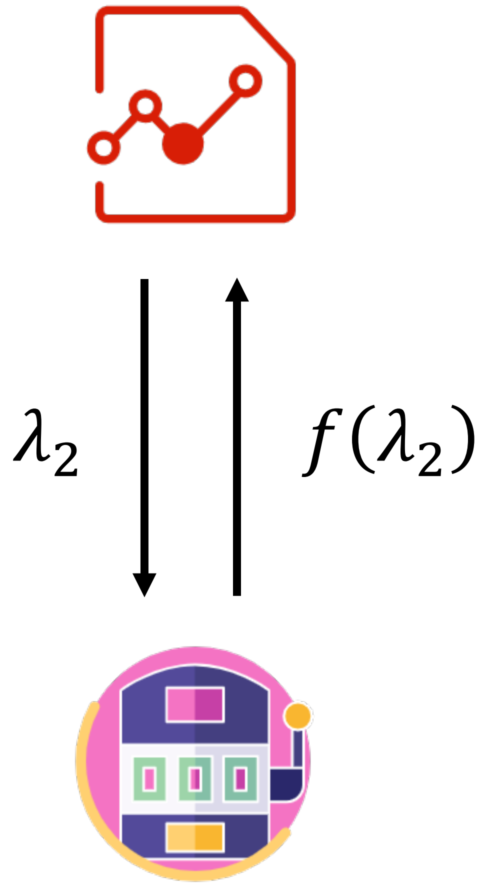
HPO: A Black-box Optimization View

❑ Problem definition

$$\min_{\lambda \in \Lambda_1 \times \cdots \times \Lambda_K} f(\lambda)$$

❑ Black-box function f

- Domain
  - i.e., search space
  - e.g., learning rate $\in \Lambda_1 = [0.001, 0.1]$
  - e.g., batch size $\in \Lambda_2 = \{16, 32, 64\}$
- Function evaluation
  - Executing the corresponding algorithm with the given hyperparameter configuration λ
  - Producing the output, e.g., validation loss



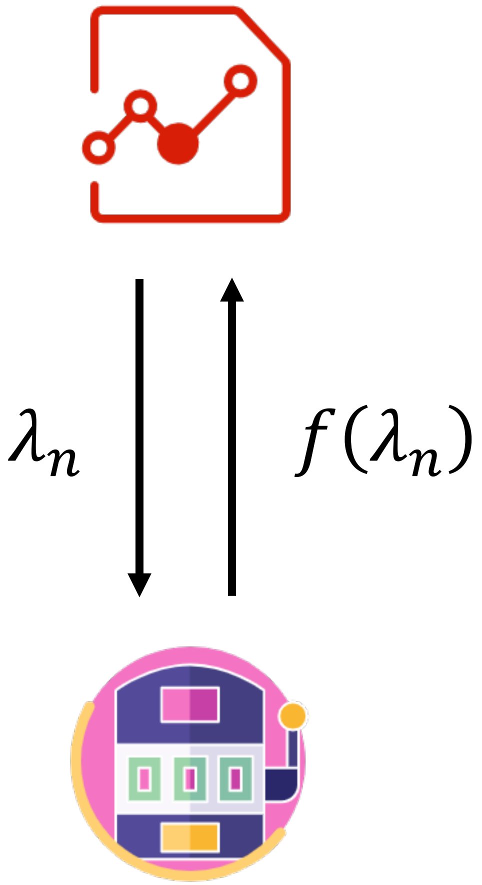
HPO: A Black-box Optimization View

❑ Problem definition

$$\min_{\lambda \in \Lambda_1 \times \cdots \times \Lambda_K} f(\lambda)$$

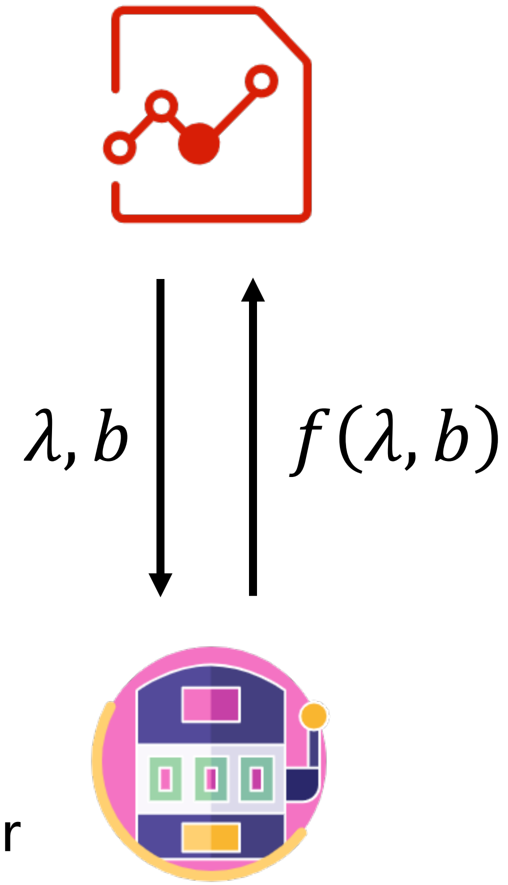
❑ Black-box function f

- Domain
  - i.e., search space
  - e.g., learning rate $\in \Lambda_1 = [0.001, 0.1]$
  - e.g., batch size $\in \Lambda_2 = \{16, 32, 64\}$
- Function evaluation
  - Executing the corresponding algorithm with the given hyperparameter configuration λ
  - Producing the output, e.g., validation loss



HPO: Multi-fidelity Strategy

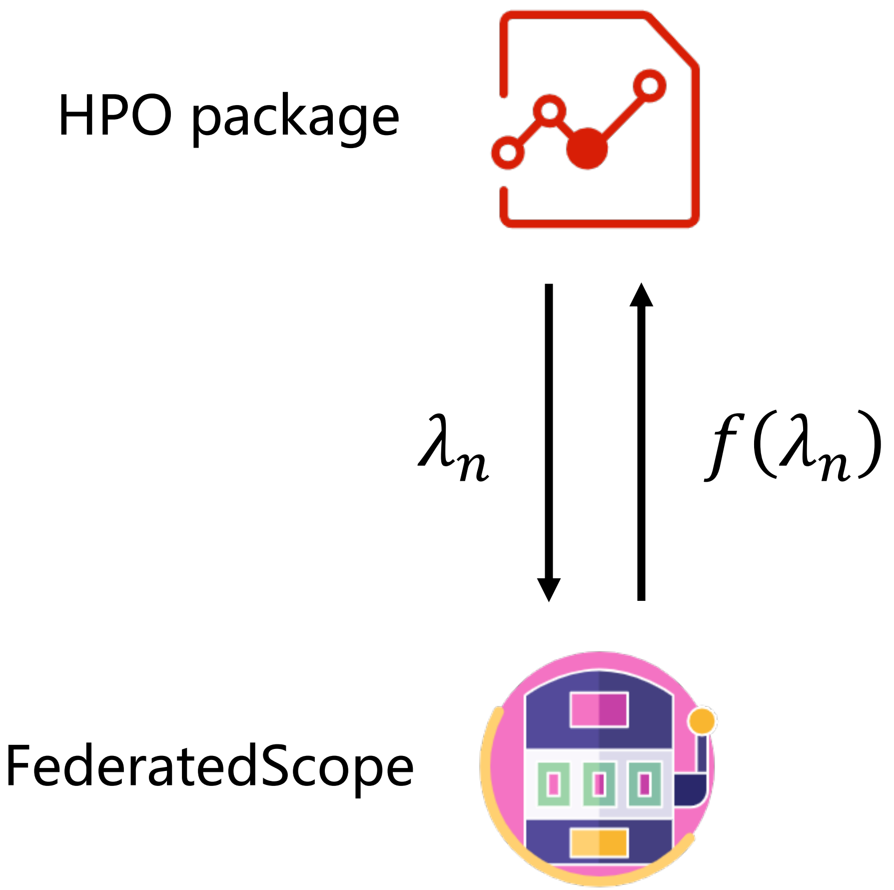
- ❑ Exact function evaluation is unaffordable
- ❑ Low-fidelity function evaluation
  - E.g., training fewer epochs, on a subset, etc.
  - Balance precision and efficiency
- ❑ Function evaluation $f(\lambda, b)$
  - Fidelity domain $b \in B_1 \times \cdots \times B_L$
    - e.g., #epoch $\in \mathrm{B}_1 = [50, 500]$
    - e.g., fraction of used training set $\in \mathrm{B}_2 = \{25\%, 50\%, 100\%\}$
  - Executing the algorithm with the given hyperparameter configuration λ and fidelity configuration b



Study FedHPO with FederatedScope

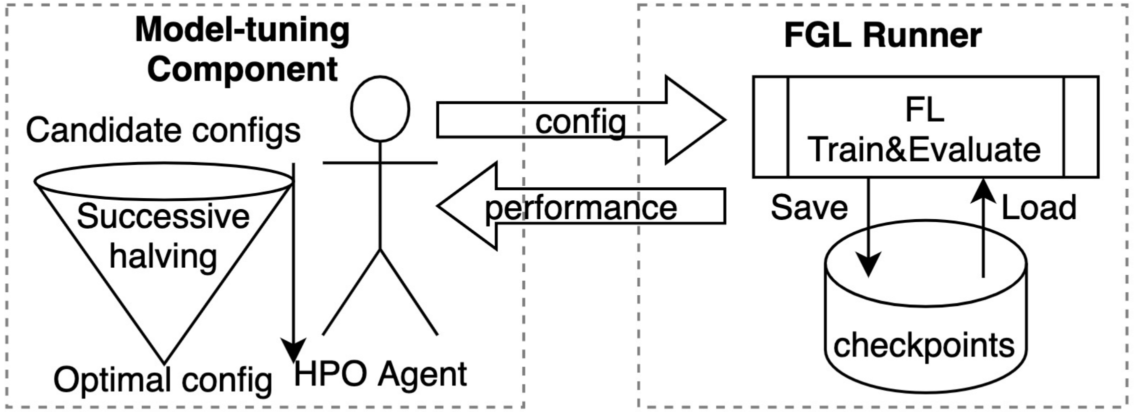
- **Apply existing HPO package to FederatedScope**
  - **Compatible and easy-to-use**
- Implement and apply multi-fidelity HPO
  - Successive halving algorithm (SHA) [1]
- Implement and apply FedHPO methods
  - FedEx [2] and FedEx wrapped by SHA
- FedHPO benchmark
  - Design, Features, and Usage of FedHPO-B [3]

Encapsulation of FederatedScope



Users instantiate an HPO agent provided by the HPO toolkit to seek optimal config

Each trial instantiates an FL runner provided by FederatedScope to generate feedback



Utilizing Emukit

```python
def exec_fl_algo(x):
    init_cfg = global_cfg.clone()
    init_cfg["train.optimizer.lr"] = x

    data = get_data(config=init_cfg)
    runner = FedRunner(data=data,
                       server_class=get_server_cls(init_cfg),
                       client_class=get_client_cls(init_cfg),
                       config=init_cfg)
    results = runner.run()

    return [results['weighted_avg']['val_loss']]

def target_function(x):
    return np.asarray([exec_fl_algo(elem) for elem in x])

space = ParameterSpace([ContinuousParameter('lr', 1e-4, .75)])
X_init = np.array([[0.005], [0.05], [0.5]])
Y_init = target_function(X_init)
bo = GPBayesianOptimization(variables_list=space.parameters,
                            X=X_init,
                            Y=Y_init)
bo.run_optimization(target_function, 50)
```

Declare the target function, i.e., executing the FL algorithm with a given learning rate

Declare search space

Apply Gaussian Process (GP) model [4]

Utilizing SMAC

```python
def exec_fl_algo(x):
    init_cfg = global_cfg.clone()
    init_cfg["optimizer.lr"] = x['lr']
    init_cfg["optimizer.weight_decay"] = x['wd']
    init_cfg["model.dropout"] = x['dropout']

    data = get_data(config=init_cfg)
    runner = FedRunner(data=data,
                       server_class=get_server_cls(init_cfg),
                       client_class=get_client_cls(init_cfg),
                       config=init_cfg)
    results = runner.run()

    return results['weighted_avg']['val_loss']
```

Declare the target function, i.e., executing the FL algorithm with a given learning rate, weight decay coefficient, dropout rate.

Utilizing SMAC

```
searchspace = ConfigurationSpace()
searchspace.add_hyperparameter(
    UniformFloatHyperparameter("lr", lower=1e-4, upper=1.0,
log=True))
searchspace.add_hyperparameter(
    UniformFloatHyperparameter("dropout", lower=.0, upper=.5))
searchspace.add_hyperparameter(
    CategoricalHyperparameter("wd", choices=[0.0, 0.5]))

scenario = Scenario({
    "runcount-limit": 50,
    "cs": searchspace,
    'output_dir': "smac_rf",
})
smac = SMAC4HPO(scenario=scenario, tae_runner=exec_fl_algo)
best_found_config = smac.optimize()
```

Declare search space

Bayesian optimization with random forest model [5]

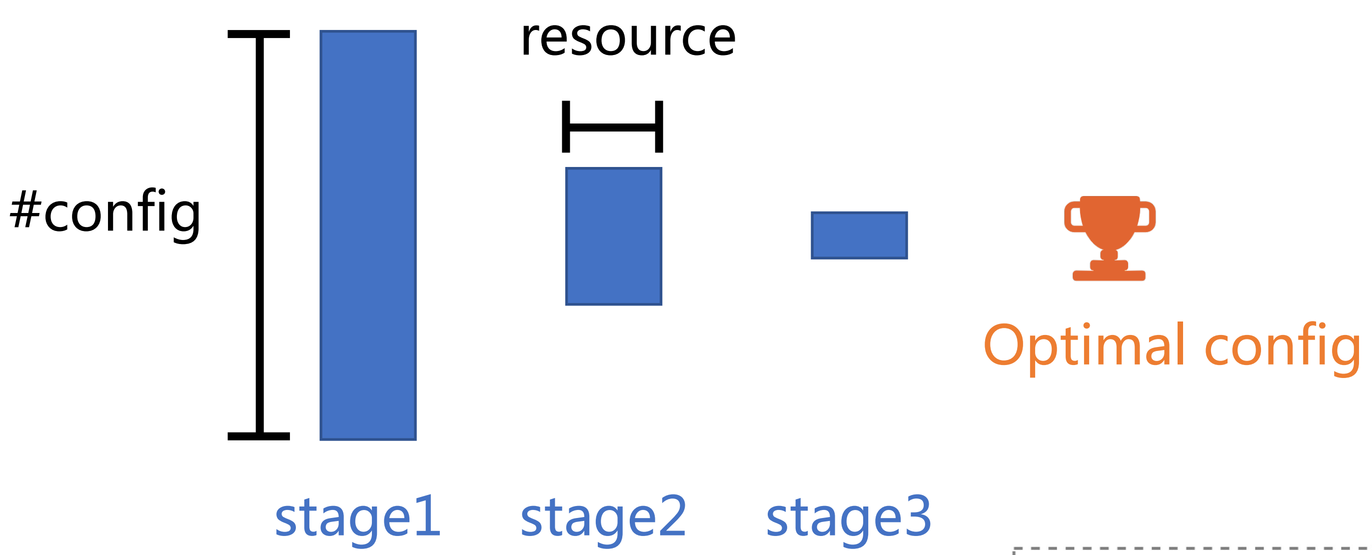
Study FedHPO with FederatedScope

- ❑ Apply existing HPO package to FederatedScope
  - Compatible and easy-to-use
- ❑ **Implement and apply multi-fidelity HPO**
  - **Successive halving algorithm (SHA) [1]**
- ❑ Implement and apply FedHPO methods
  - FedEx [2] and FedEx wrapped by SHA
- ❑ FedHPO benchmark
  - Design, Features, and Usage of FedHPO-B [3]

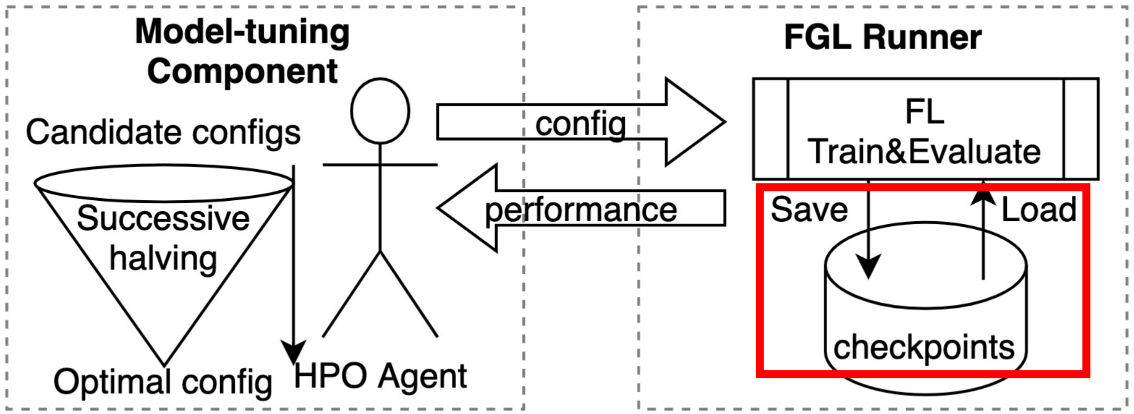
Successive Halving Algorithm for HPO

- Iterative procedure
  - Initiate with n candidate configs
  - In i-th stage, evaluate each config with r_i resource
  - The best $\frac{1}{\eta}$ candidate configs are promoted into the next stage
  - Repeat these steps until only one config remaining

Motivation of SHA



More promising configs are evaluated with more resources!



Implementation in FederatedScope

```
class IterativeScheduler(ModelFreeBase):
  def _iteration(self, configs):
    perfs = self._evaluate(configs)
    return perfs

  def optimize(self):
        current_configs = deepcopy(self._init_configs)
        last_results = None
        while not self._stop_criterion(current_configs, last_results):
            current_perfs = self._iteration(current_configs)
            last_results = summarize_hpo_results(
                current_configs,
                current_perfs)
            self._stage += 1
            logger.info("Stage{}".format(self._stage))
            logger.info(last_results)
            current_configs = self._generate_next_population(
                current_configs, current_perfs)

        return current_configs
```

Evaluate all candidates of current stage

Update the candidates w.r.t. current evaluation results

Implementation in FederatedScope

```
class SuccessiveHalvingAlgo(IterativeScheduler):

  def _stop_criterion(self, configs, last_results):
      return len(configs) <= 1

  def _generate_next_population(self, configs, perfs):
      next_population = topK(configs, perfs, len(configs)*self._eta)

      for trial_cfg in next_population:
          trial_cfg['federate.restore_from'] = trial_cfg[
                'federate.save_to']

      return next_population
```

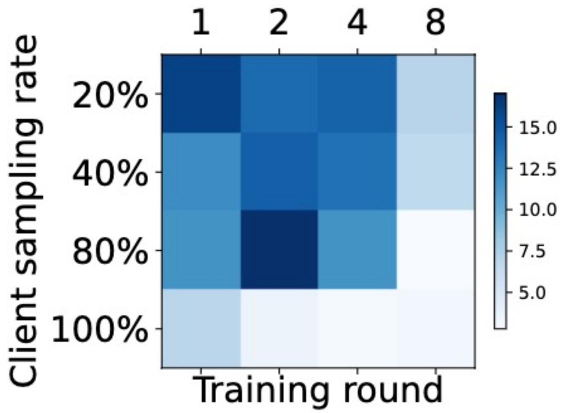
In PBT [6], replace this by checking improvements of performances

Let the survived candidates start FL course from their corresponding latest checkpoints

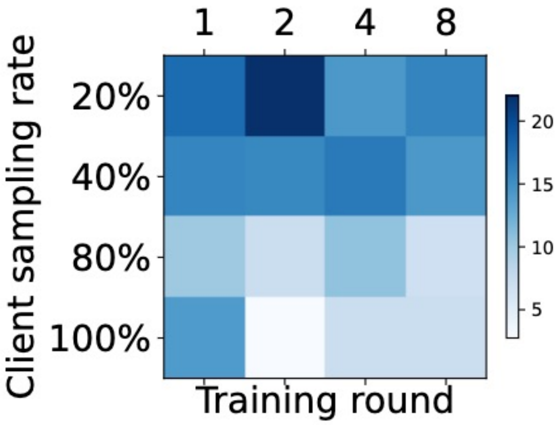
Choice of Fidelity Dimension in SHA

❑ Training rounds v.s. Client sampling rate

- Train GNN by FedAvg on PubMed
- Apply SHA to optimize the hyperparameters
- The rank of searched config's test accuracy (the smaller, the better)



(a) GCN.



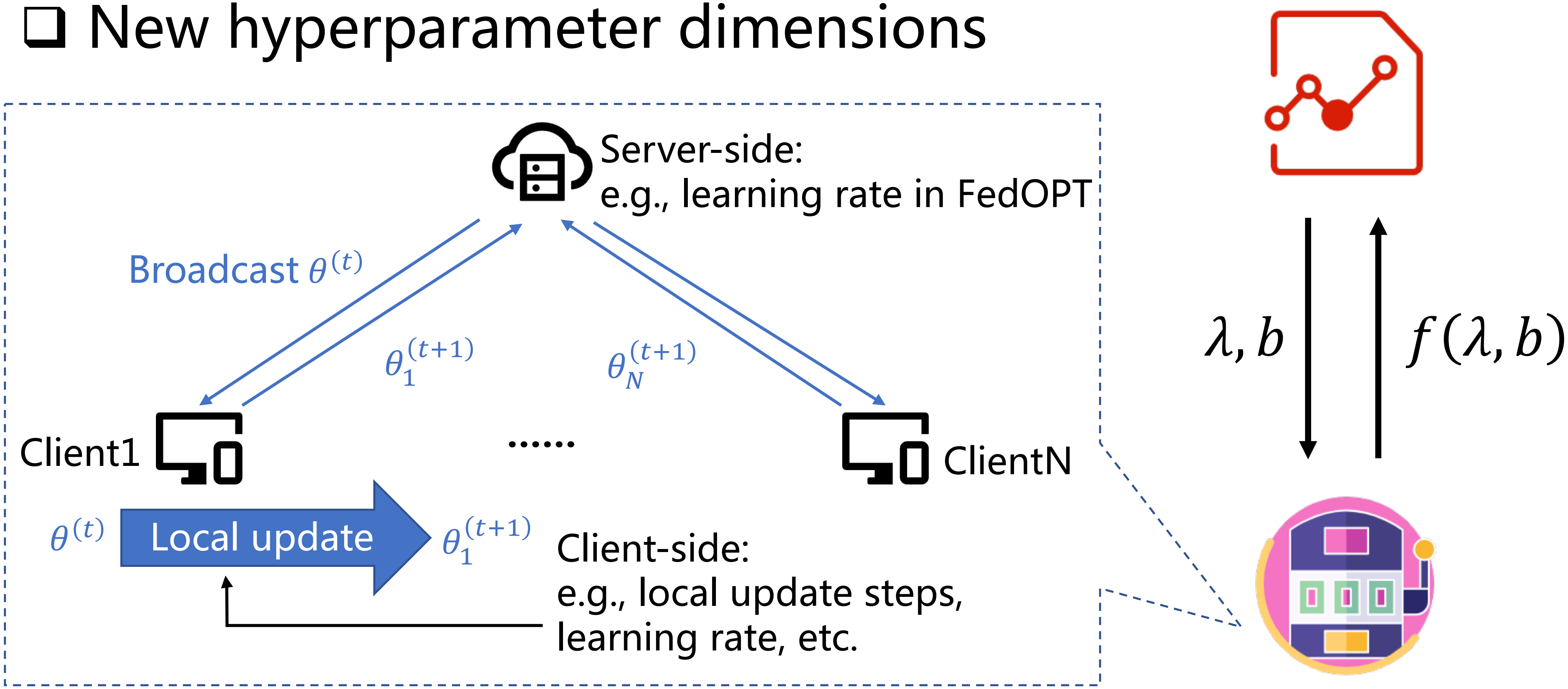
(b) GPR-GNN.

Study FedHPO with FederatedScope

- Apply existing HPO package to FederatedScope
  - Compatible and easy-to-use
- Implement and apply multi-fidelity HPO
  - Successive halving algorithm (SHA) [1]
- **Implement and apply FedHPO methods**
  - **FedEx [2] and FedEx wrapped by SHA**
- FedHPO benchmark
  - Design, Features, and Usage of FedHPO-B [3]

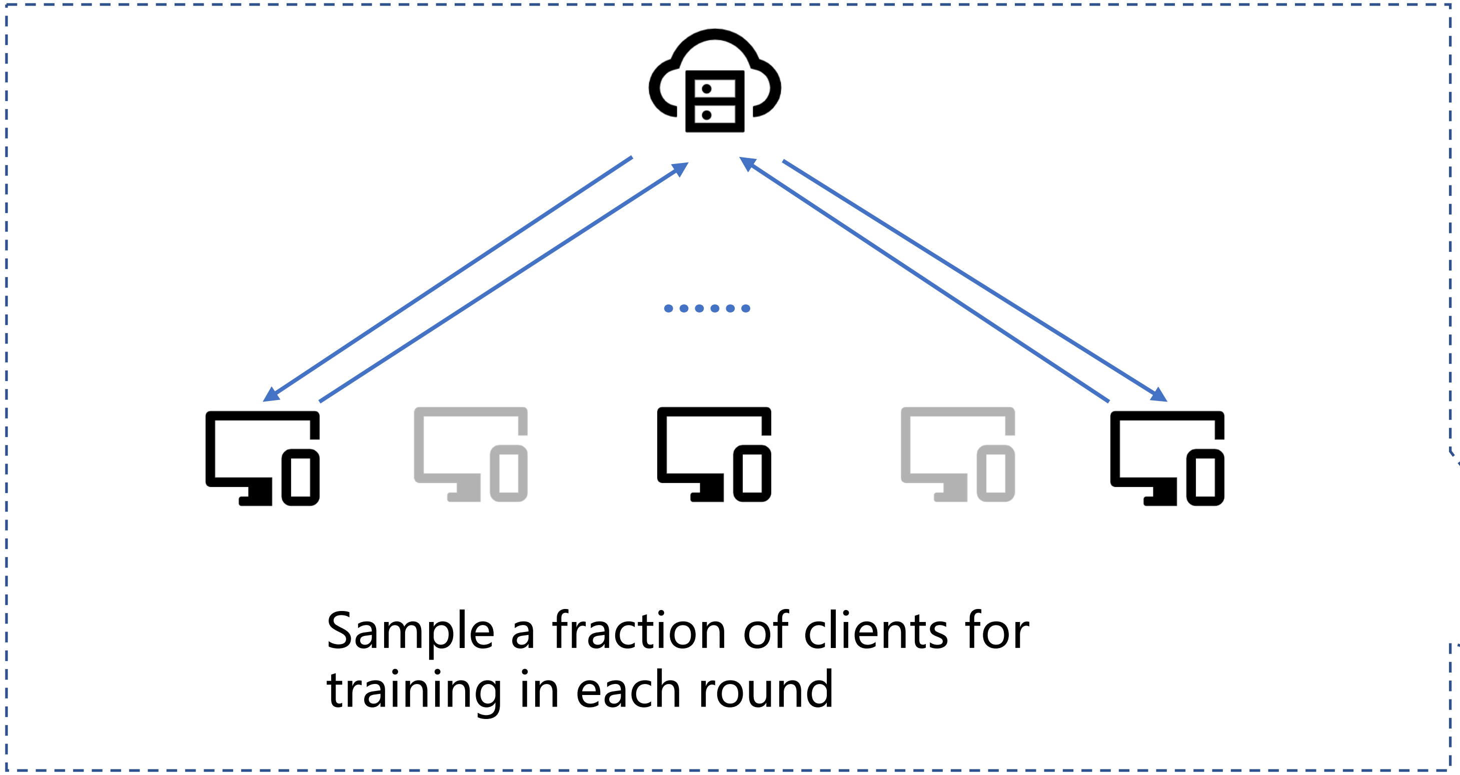
Uniqueness of FedHPO

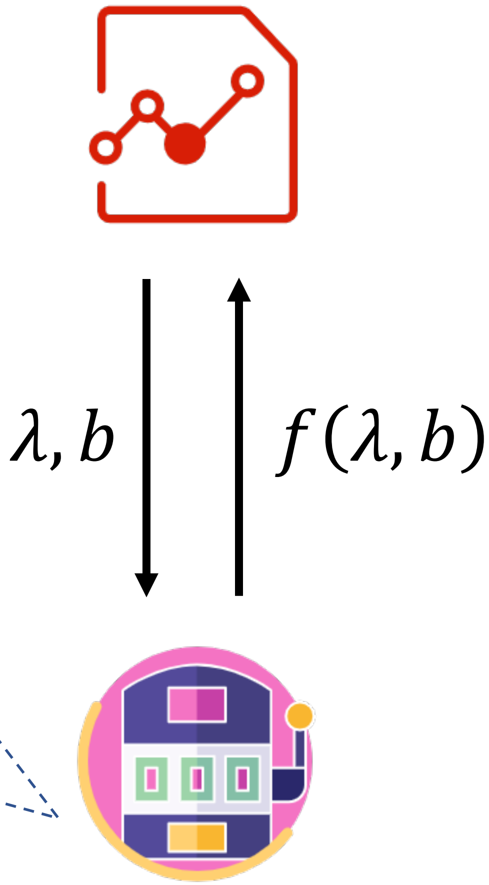
- ❑ New hyperparameter dimensions



Uniqueness of FedHPO

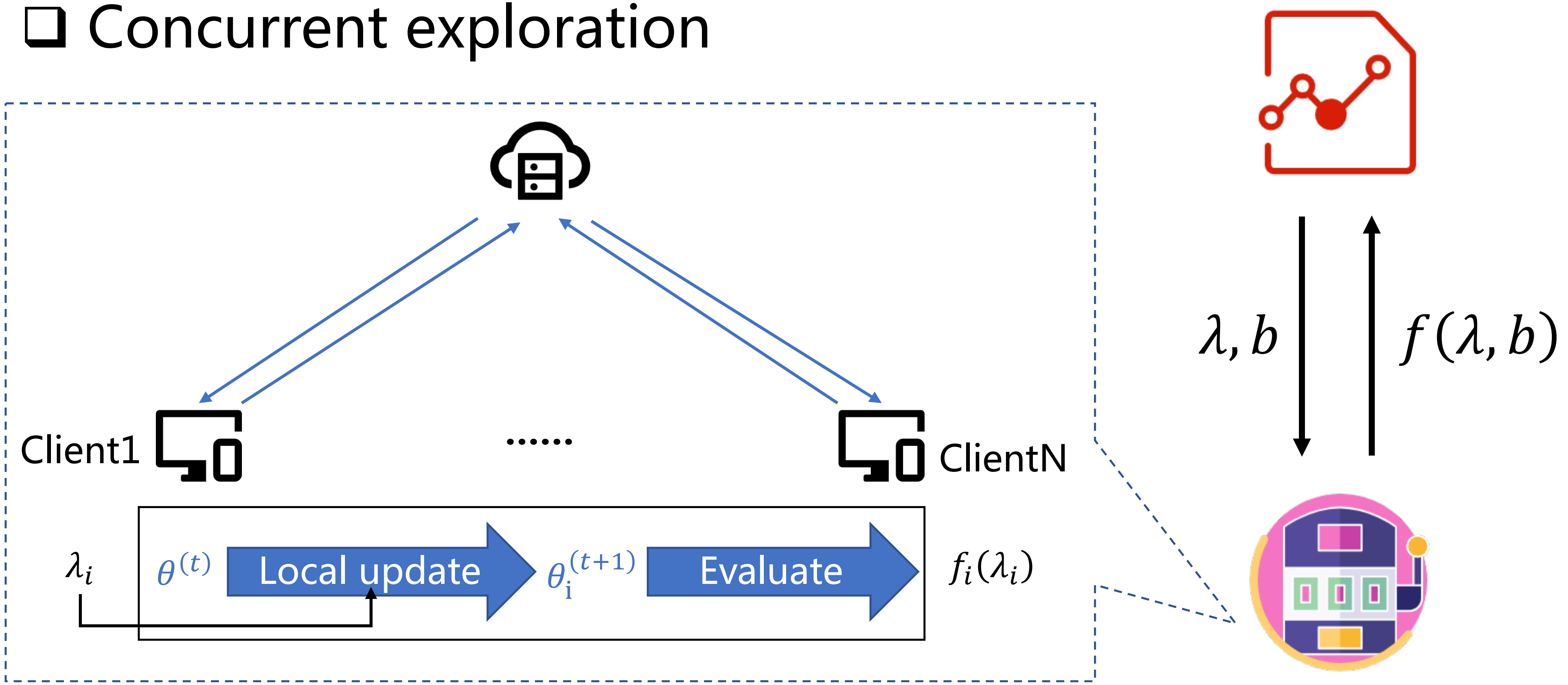
❑ New fidelity dimensions





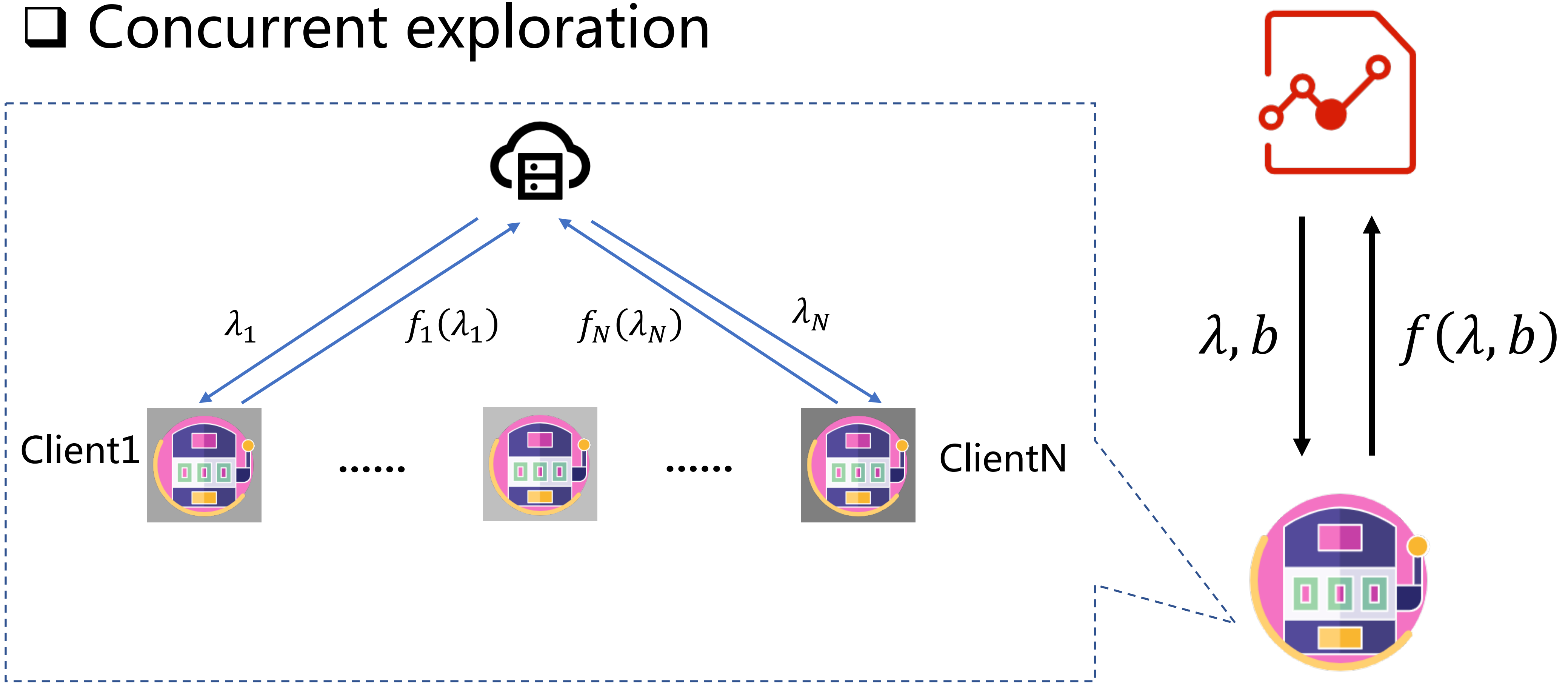
Uniqueness of FedHPO

- ❑ Concurrent exploration



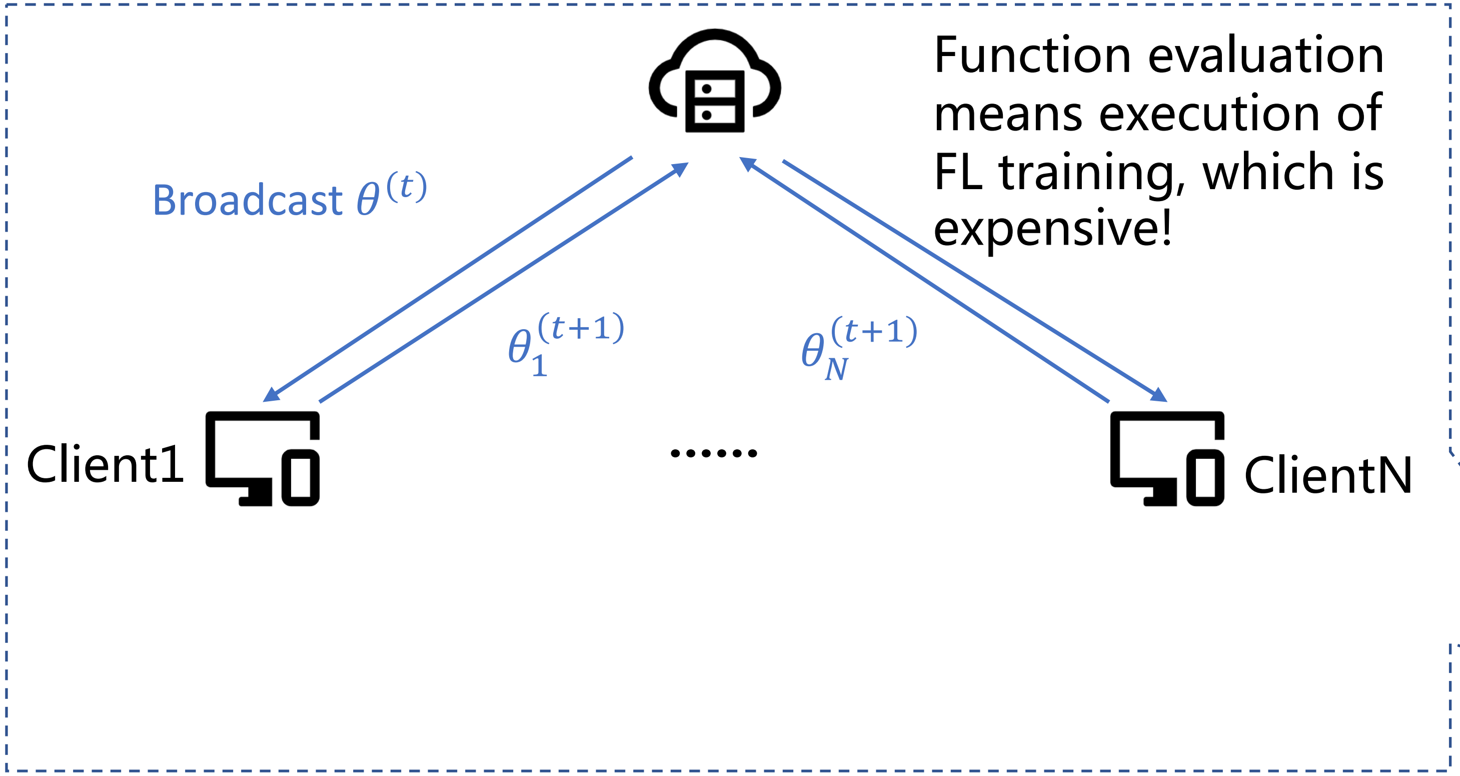
Uniqueness of FedHPO

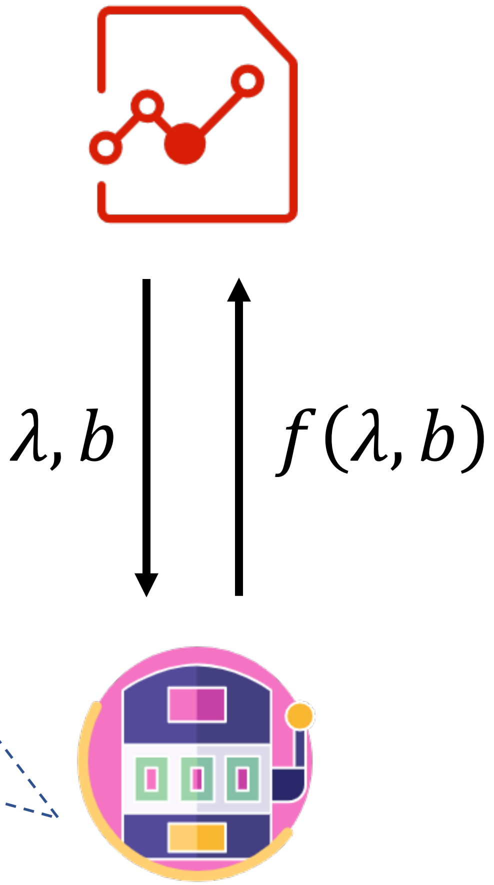
- ❑ Concurrent exploration



Uniqueness of FedHPO

- ❑ One-shot optimization





FedHPO methods

❑ FedEx: a very recent and representative work

- Make concurrent exploration for client-side hyperparameters
- Can be wrapped by traditional HPO methods, e.g., SHA
  - For optimizing the server-side hyperparameters
  - For optimizing the hyperparameters of FedEx

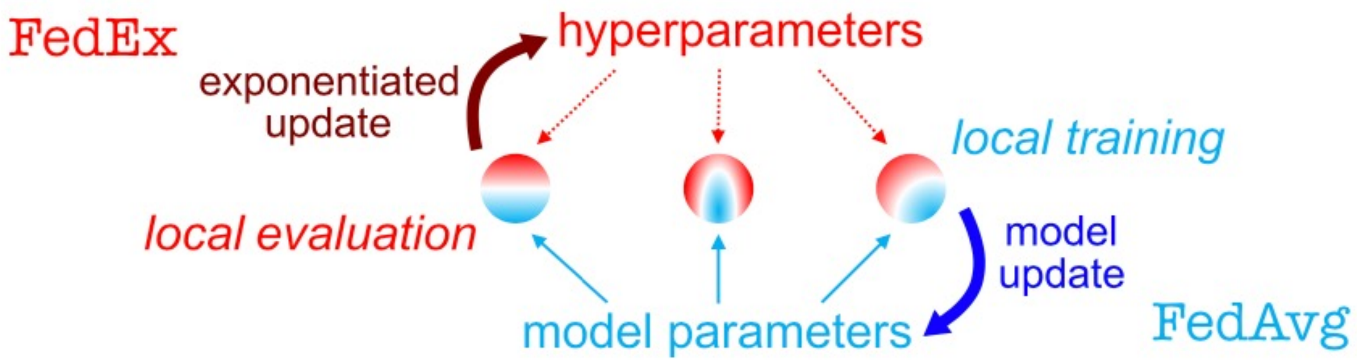


Image source: Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. In NeurIPS 2021.

Implementation in FederatedScope

❑ FedEx: a very recent and representative work

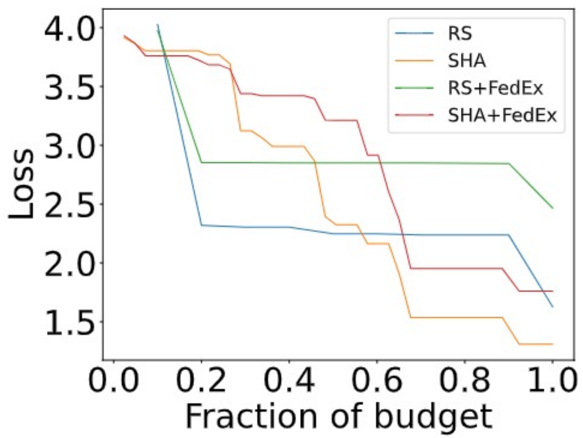
- Introducing additional behaviors for server and client classes
- Allowing trainer class to make validation during local updates

```
class Server(object):
    def handler_for_receiving_updates(args):
        ... ...
        if config.apply_fedex == True:
            # Choose hyperparameters for the client
            cfg = sample_cfg(cfg_candidates, args.hpo_feedback)
        # Continue to handling the message accordingly
        ... ...
```

```
class Client(object):
    def handler_for_receiving_models(args):
        ... ...
        if config.apply_fedex == True:
            # Apply the received hyperparameters
            trainer.apply_cfg(args.received_config)
        # Continue to handling the message accordingly
        ... ...
```

Does Concurrent Exploration work?

❑ Comparing Wrapped FedEx to the corresponding wrapper



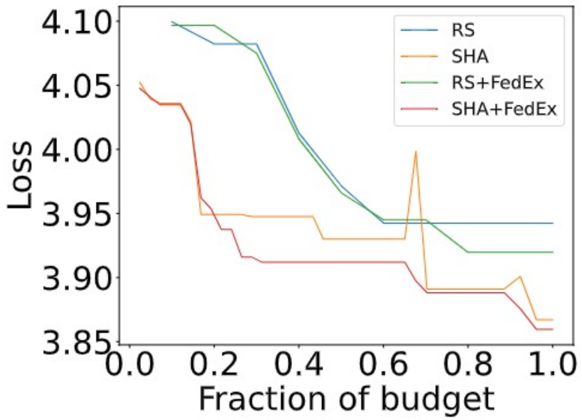


Figure 5: Mean validation loss over time. **Left**: FedAvg. **Right**: FedOPT.

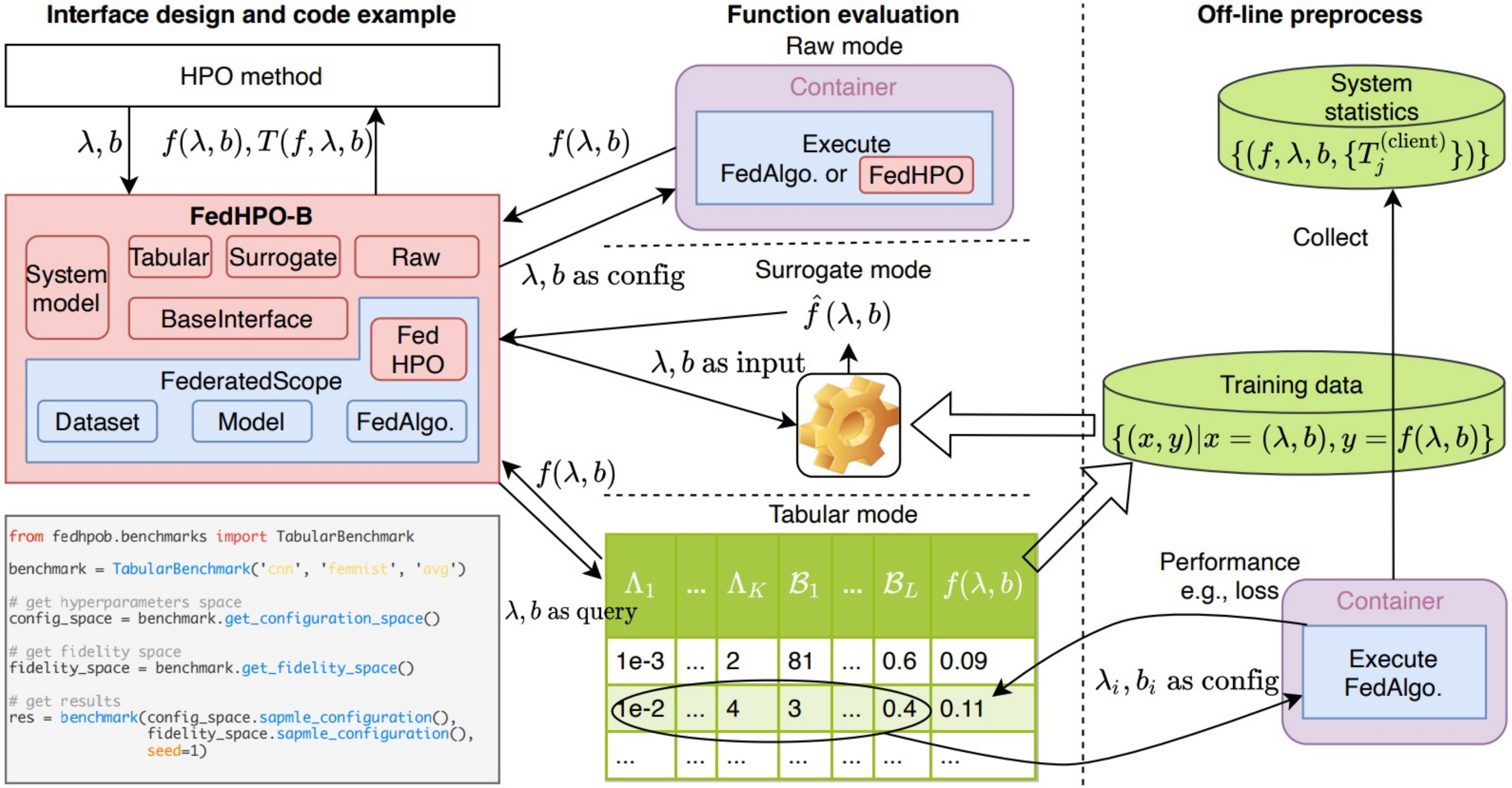
Methods	Test Accuracy
RS	67.14 ± 8.46
SHA	75.15 ± 3.44
RS+FedEx	71.25 ± 8.79
SHA+FedEx	**77.46 ± 1.78**

Table 5: Evaluation about the searched configurations: Mean test accuracy (%) ± standard deviation.

Study FedHPO with FederatedScope

- Apply existing HPO package to FederatedScope
  - Compatible and easy-to-use
- Implement and apply multi-fidelity HPO
  - Successive halving algorithm (SHA)
- Implement and apply FedHPO methods
  - FedEx and FedEx wrapped by SHA
- **FedHPO benchmark**
  - **Design, Features, and Usage of FedHPO-B**

FedHPO-B: FedHPO Benchmark Suite [3]



FedHPO-B: Efficiency

- Tabular mode
  - Evaluating functions by looking up tables
  - Viable only for discrete search space
- Surrogate mode
  - Evaluating functions via model inference
  - Accuracy of $\hat{f}(\lambda, b)$ matters
- Raw mode
  - Standalone simulation avoids communication
  - Execution time is meaningless

Common issue: how to acquire the execution time of function evaluation

FedHPO-B: Efficiency

❑ System model

- Configurable
- Model parameters collected from realistic scenarios are provided

$$T(f, \lambda, b) = T_{\text{comm}}(f, \lambda, b) + T_{\text{comp}}(f, \lambda, b),$$

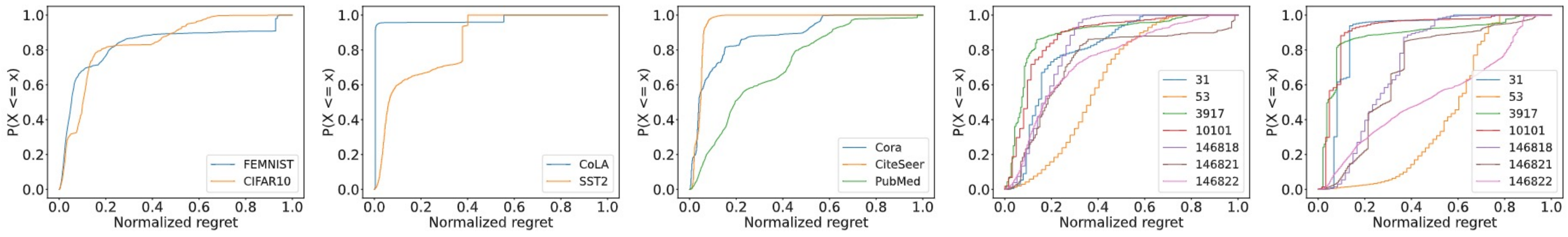
$$T_{\text{comm}}(f, \lambda, b) = \max\left(\frac{N \times S_{\text{down}}(f, \lambda)}{B_{\text{up}}^{(\text{server})}}, \frac{S_{\text{down}}(f, \lambda)}{B_{\text{down}}^{(\text{client})}}\right) + \frac{S_{\text{up}}(f, \lambda)}{B_{\text{up}}^{(\text{client})}},$$

$$T_{\text{comp}}(f, \lambda, b) = \mathbb{E}_{T_i^{(\text{client})} \sim \text{Exp}(\cdot | \frac{1}{c(f, \lambda, b)}), i=1,\ldots,N}[\max(\{T_i^{(\text{client})}\})] + T^{(\text{server})}(f, \lambda, b),$$

FedHPO-B: Comprehensiveness

- ❑ FedHPO-B provides various FedHPO tasks

Model	#Dataset	Domain	#Client	#FL Algo.	#Cont.	#Disc.	Opt. budget
CNN	2	CV	200	2	4	2	20 days
BERT [7]	2	NLP	5	2	4	2	20 days
GNN	3	Graph	5	2	4	1	1 days
LR	7	Tabular	5	2	3	1	21,600 seconds
MLP	7	Tabular	5	2	4	3	43,200 seconds



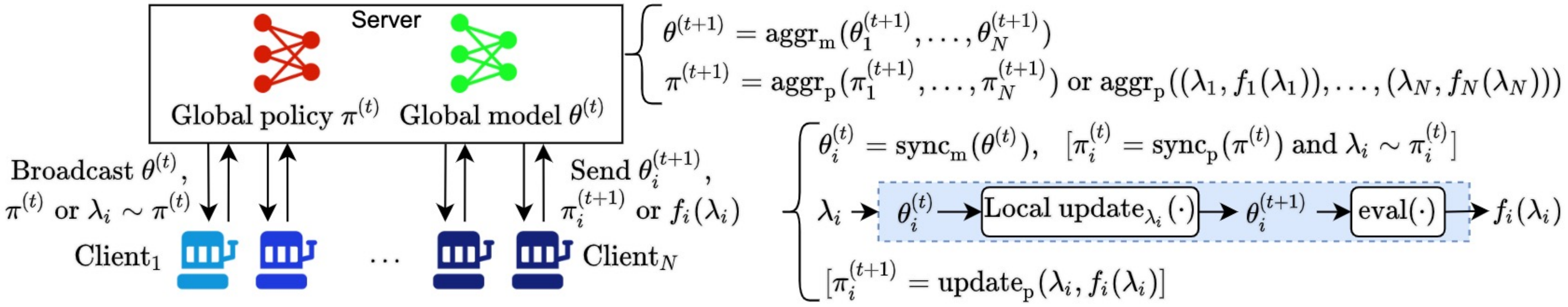
(a) CNN (b) BERT (c) GNN (d) LR (e) MLP

FedHPO-B: Extensibility

❑ FedHPO-B is developed based on FederatedScope

- Splitter: standalone dataset → federated dataset
- Event-driven framework with customizable messages

❑ A general view to unify many FedHPO methods



Empirical Study based on FedHPO-B

- ❑ How traditional HPO methods perform in the FL setting

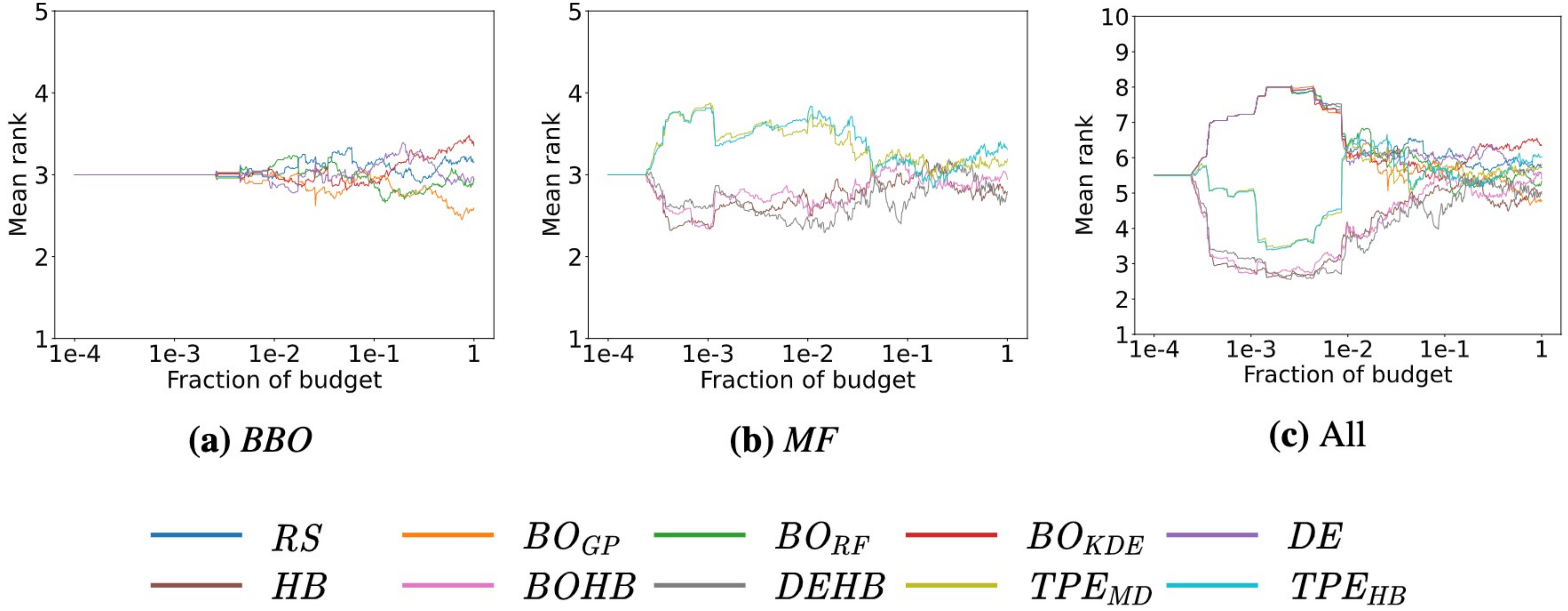


Figure 4: Mean rank over time on all FedHPO problems (with FedAvg).

Future Directions

- ❑ Personalized FedHPO
  - Non-IIDness might lead to different optimal hyperparameter configs
- ❑ Byzantine-resilient FedHPO methods
  - Malicious contribute noisy or even bad results of function evaluations
- ❑ A more realistic cross-silo setting
  - Cooperation and competition

References of FedHPO

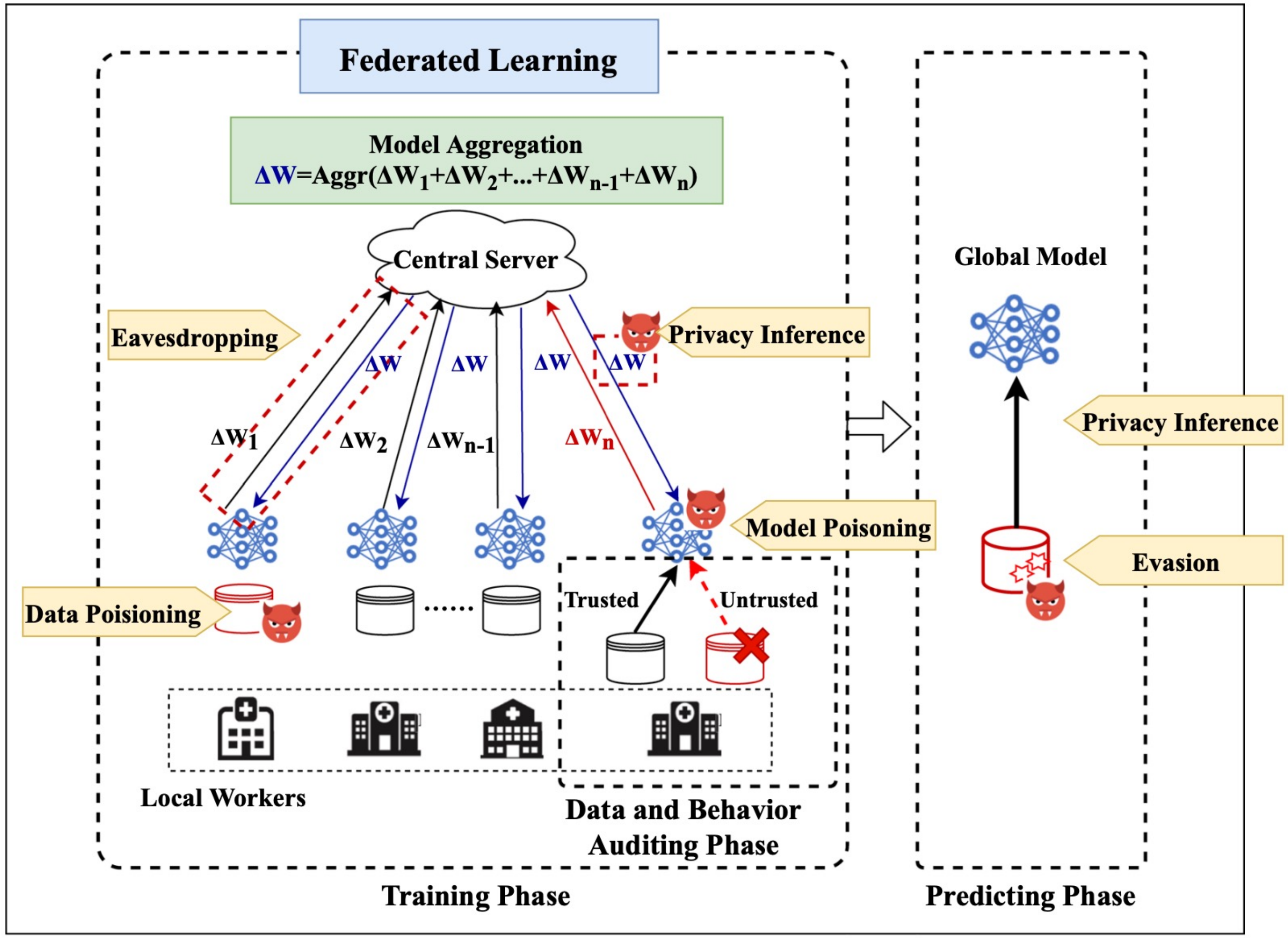
[1] Hyperband: A novel bandit-based approach to hyperparameter optimization. In JMLR 2017.
[2] Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. In NeurIPS 2021.
[3] FedHPO-B: A Benchmark Suite for Federated Hyperparameter Optimization. In arXiv 2022.
[4] SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. In J. Mach. Learn. Res. 2022.
[5] Sequential model-based optimization for general algorithm configuration. In ICLIO 2011.
[6] Population Based Training of Neural Networks. In arXiv 2017.

Agenda of Tutorial

- Overview
- Personalized Federated Learning
- Federated Graph Learning
- Federated Hyperparameter Optimization
- Privacy Attacks

Privacy Attacks

Threats in Federated Learning



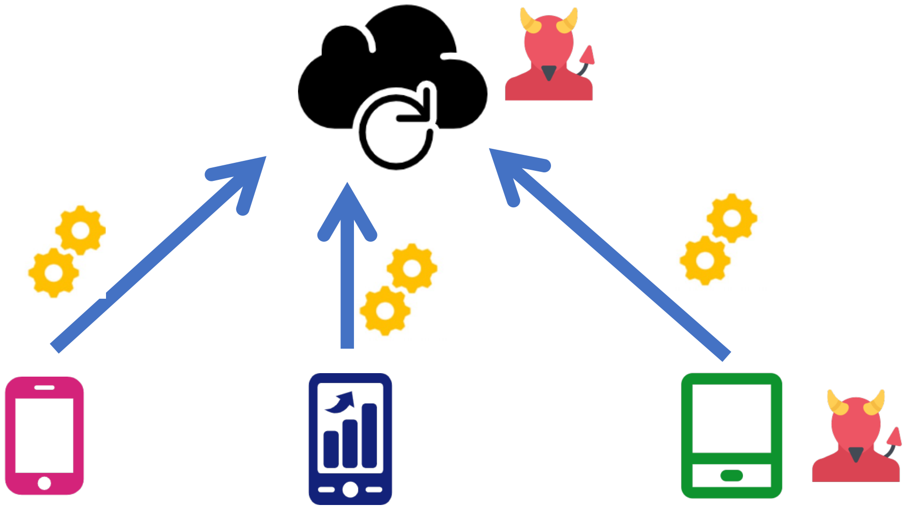
[1] Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives. Cybersecur. 2022.

Privacy Attack

☆ **Preventing the privacy leakage is one of the important requirements of FL!**

Privacy Attack:

- From the learning course of FL, infer the sensitive information related to clients' private data



- Understanding and applying privacy attacks on FL is an effective and intuitive way to detect/prevent the privacy leakage during FL course!
- Other attacks are also important but not FL-specific. More works needed…

Overview of Privacy Attack

Privacy Attack

Action Type

- Active Attack
- Passive Attack

Target

- Membership Inference Attack
- Property Inference Attack
- Class Representative Attack
- Training Data/Label Inference

Overview of Privacy Attack

Privacy Attack

Action Type

- Active Attack
- Passive Attack

Target

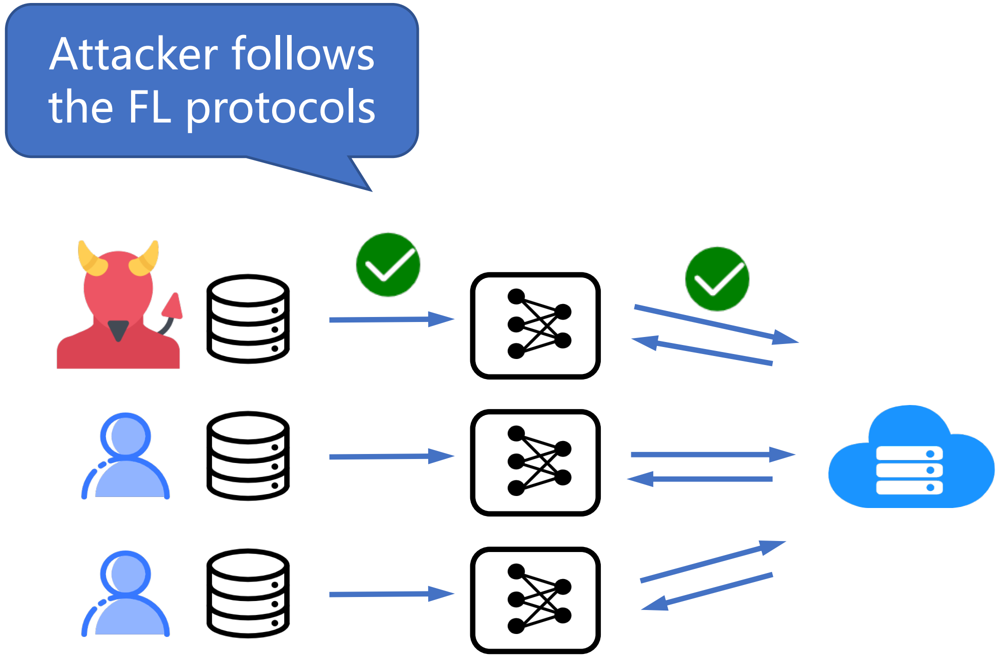
- Membership Inference Attack
- Property Inference Attack
- Class Representative Attack
- Training Data/Label Inference

Attack Action Type

Passive Attack:

Attacker:

- Honest-but-curious
- Steal the private information meanwhile not violating the FL protocols



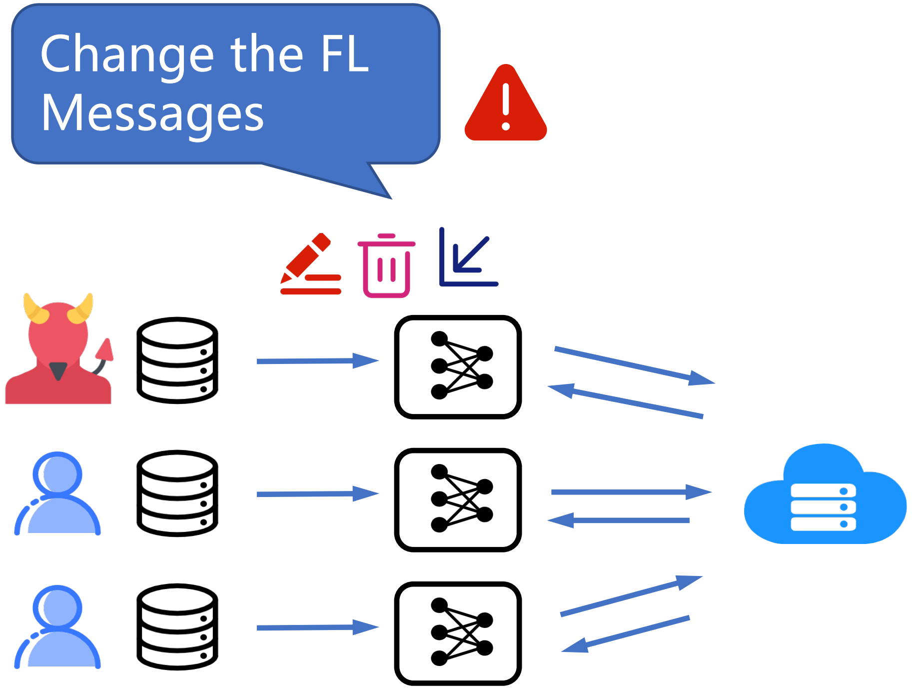
Invisible!
Hard to Detect!

Attack Action Type

Active Attack:

Attacker:

- Malicious
- Steal the private information by violating the FL protocols
  - Change loss function, gradients, model updates, training data



Overview of Privacy Attack

Privacy Attack

Action Type

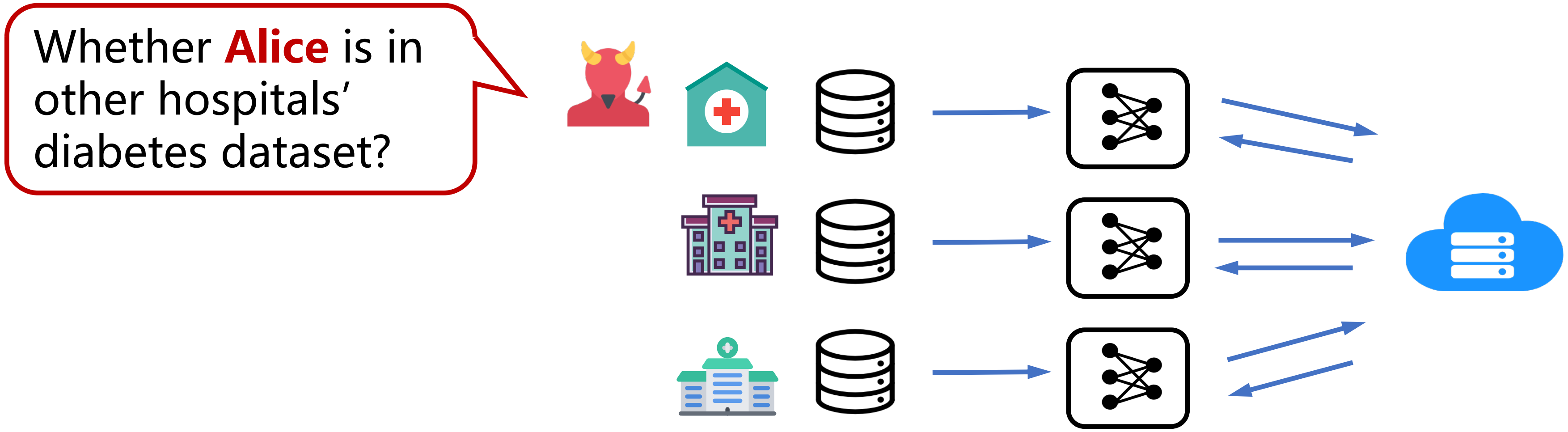
- Active Attack
- Passive Attack

Target

- Membership Inference Attack
- Property Inference Attack
- Class Representative Attack
- Training Data/Label Inference

Membership Inference Attack

- **Goal:** Infer whether a specific data instance exists in other clients' private datasets
- **Attacker Role:** Client or Server
- Example:



Membership Inference Attack

GradAscent [2]:

Active Attack

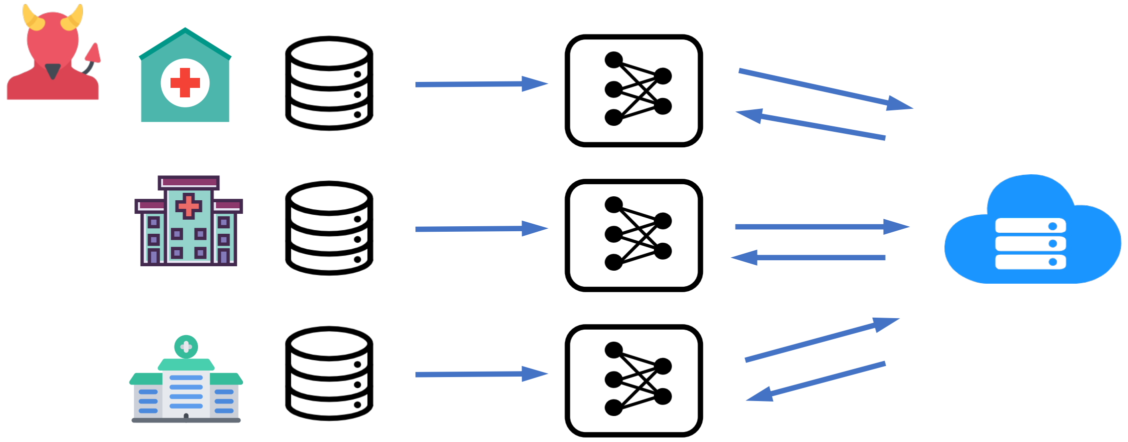
Goal: Infer whether target data x in other clients' private dataset.

At t-th round:

$$W \leftarrow W + \gamma \frac{\partial L_x}{\partial W}$$

$$W \leftarrow W - \gamma \frac{\partial L}{\partial W}$$

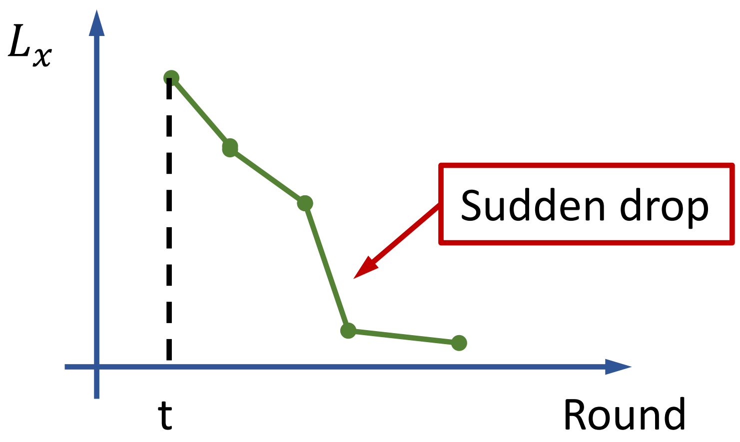
$$W \leftarrow W - \gamma \frac{\partial L}{\partial W}$$



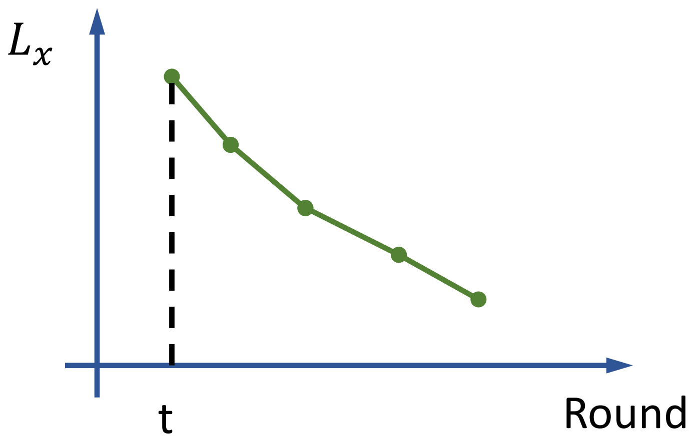
[2] Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. IEEE Symposium on Security and Privacy 2019

Membership Inference Attack

GradAscent [2]:



x in other clients' private datasets



x not in other clients' private datasets

Membership Inference Attack

Running GradAscent in FederatedScope:

- Configuration (insert these lines into the current file):

```
attack:
  attack_method: GradAscent
  attacker_id: 5
  inject_round: 0
```

Malicious attacker id

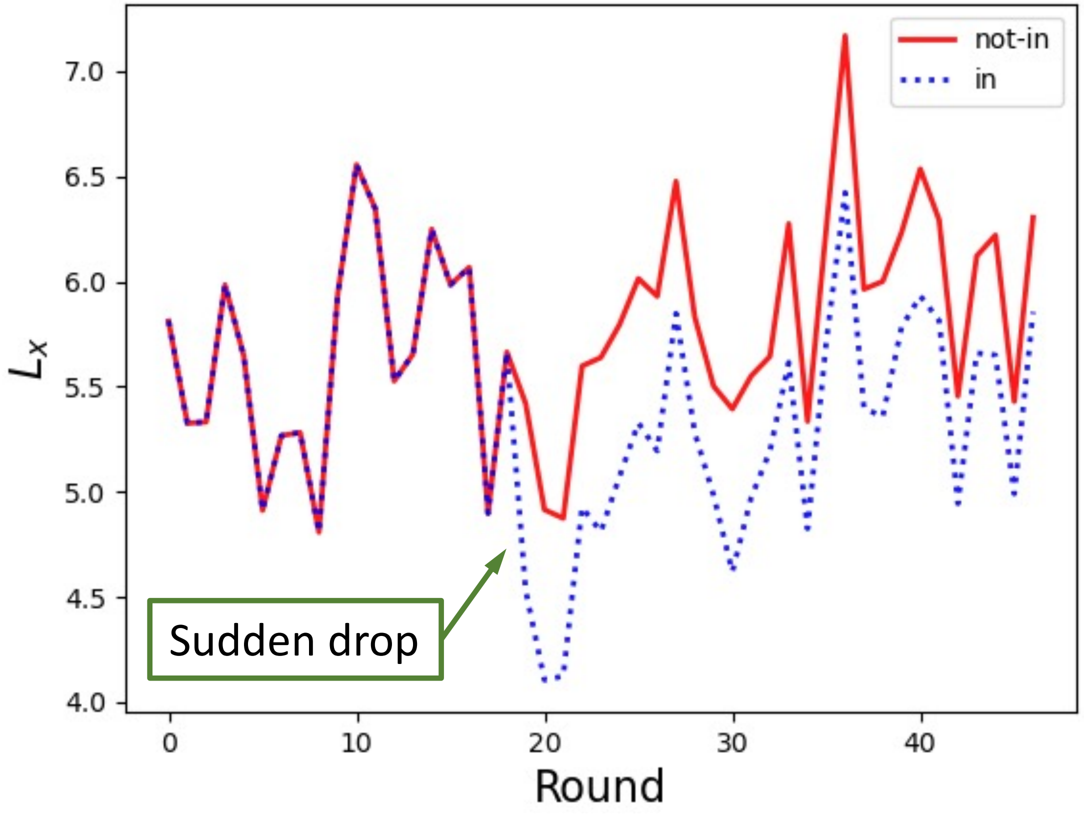
The round to run the gradient ascent on the target dataset.

- The command to run the example attack on Femnist Dataset with FedAvg:

```
python federatedscope/main.py --cfg
federatedscope/attack/example_attack_config/gradient_ascent_MIA
_on_femnist.yaml
```

Membership Inference Attack

- Running Result:



Property Inference Attack

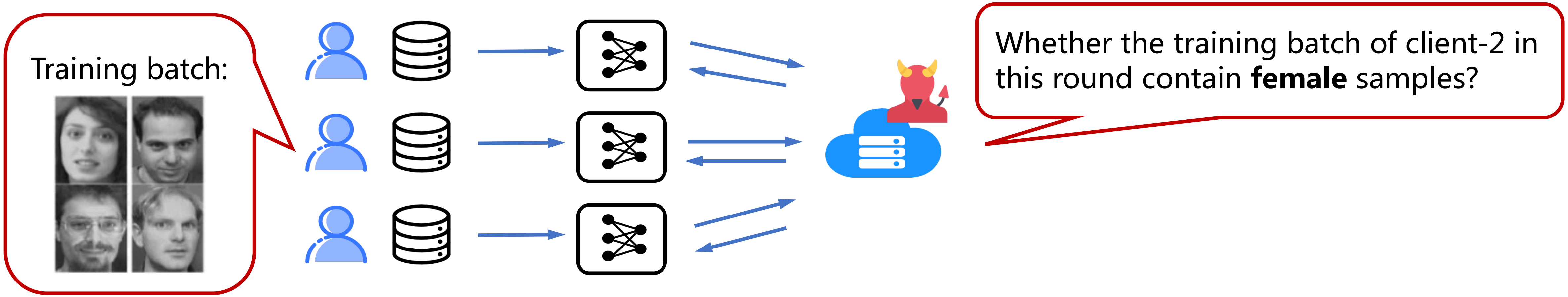
- **Goal:** Infer dataset properties
  - Properties:
    - Possibly sensitive
    - May not belong to the feature set
  - When batch size > 1:
    - Whether the training batch at this round contain a certain property value

- **Attacker Role:** Server

Property Inference Attack

Example:

- FL task: whether wearing glasses?
- Additional sensitive properties: Gender



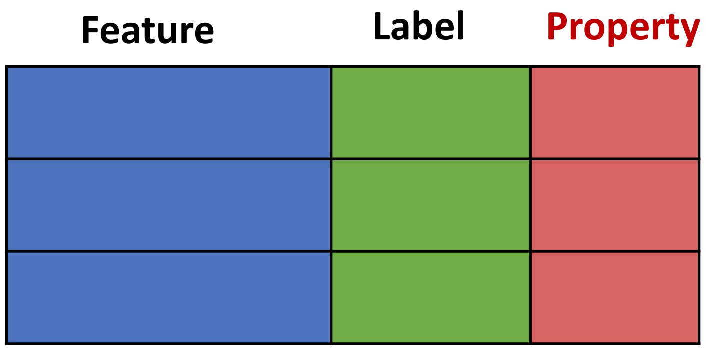
Property Inference Attack

Passive Attack

PassivePIA [2]:

Attacker:

- Role: Server
- Prior Knowledge:
  - Auxiliary dataset (to be used after each batch)



Auxiliary Dataset

[2] Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. IEEE Symposium on Security and Privacy 2019.

Property Inference Attack

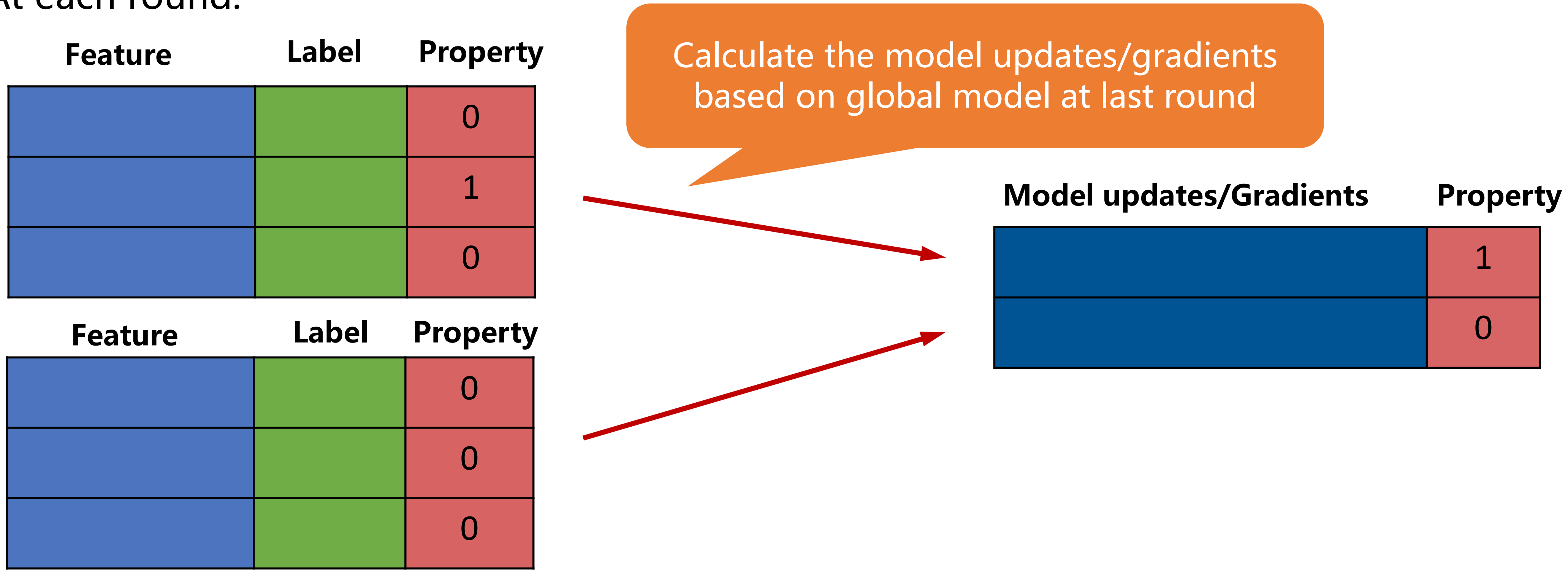
PassivePIA [2]:

- Create the training dataset for property classifier
  - Feature: Model updates/gradients
  - Label: Property
- Train the property classifier
- Infer the property on the collected model updates/gradients during FL

Property Inference Attack

PassivePIA: Create the training dataset for property classifier:

At each round:



Data batches from auxiliary data

Property Inference Attack

Running PassivePIA in FederatedScope on synthetic dataset:

- Synthetic dataset:

    - The feature x with 5 dimension is generation by $N(0, 0.5)$;
    - Label related: $w_x \sim N(0,1)$ and $b_x \sim N(0,1)$;
    - Property related: $w_p \sim N(0,1)$ and $b_p \sim N(0,1)$;
    - Label: $y = w_x x + b_x$
    - Property: $\text{prop} = \text{sigmoid}(w_p x + b_p)$

Property Inference Attack

Running PassivePIA in FederatedScope on synthetic dataset:

- Configuration (insert these lines into the current file):

```
attack:
  attack_method: PassivePIA
  classifier_PIA: svm
```

The type of property classifier

- The command to run the example attack on Synthetic Dataset with FedAvg:

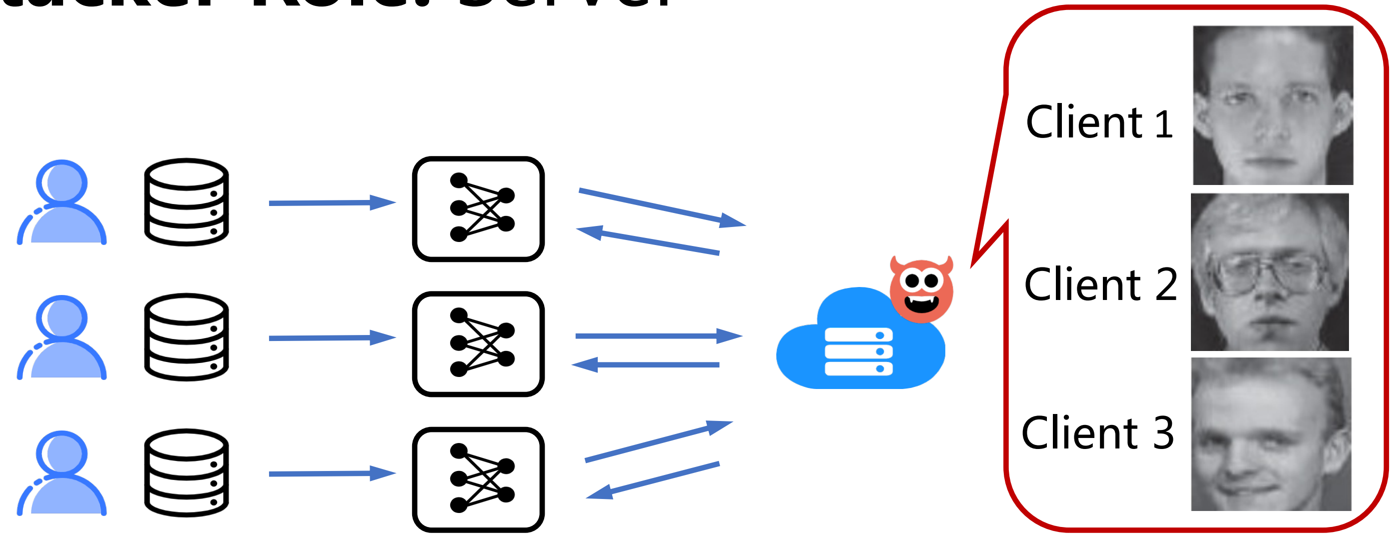
```
python federatedscope/main.py --cfg
federatedscope/attack/example_attack_config/PIA_toy.yaml
```

- Running Result:

```
(passive_PIA:167)INFO: =============== PIA accuracy on auxiliary test dataset: 1.0
```

Training Data/Label Inference Attack

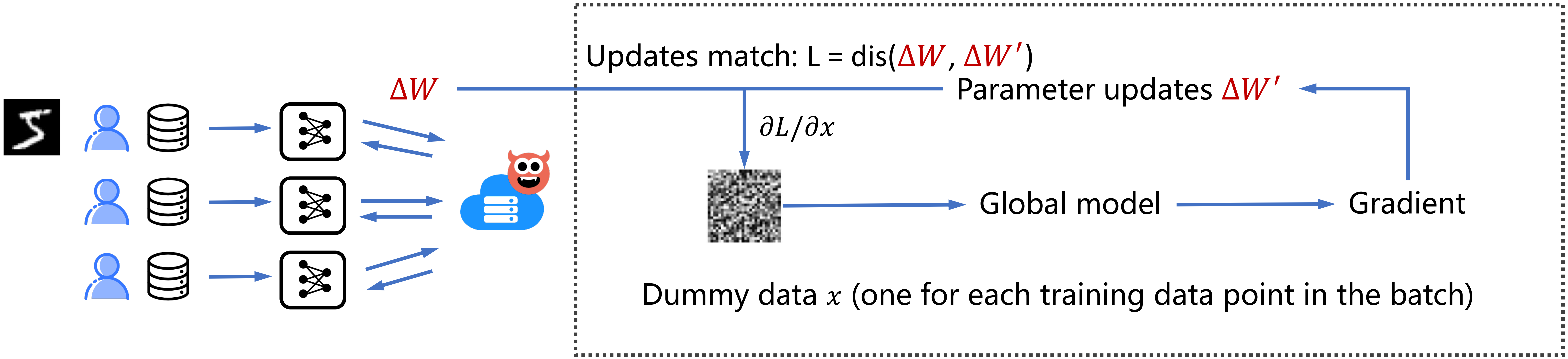
- **Goal:** Reconstruct private training samples from intermediate information transmitted during FL
- **Attacker Role:** Server



Training Data/Label Inference Attack

Passive Attack

- DLG [4], iDLG [5], InvertGradient [6]



[4] Deep leakage from gradients. NeurIPS 2019.
[5] idlg: Improved deep leakage from gradients[J]. arXiv preprint, 2020.
[6] Inverting gradients-how easy is it to break privacy in federated learning?. NeurIPS 2020.

Training Data/Label Inference Attack

Running DLG in FederatedScope:

- Configuration:

```
attack:
  attack_method: DLG
  max_ite: 500
  reconstruct_lr: 0.1
```

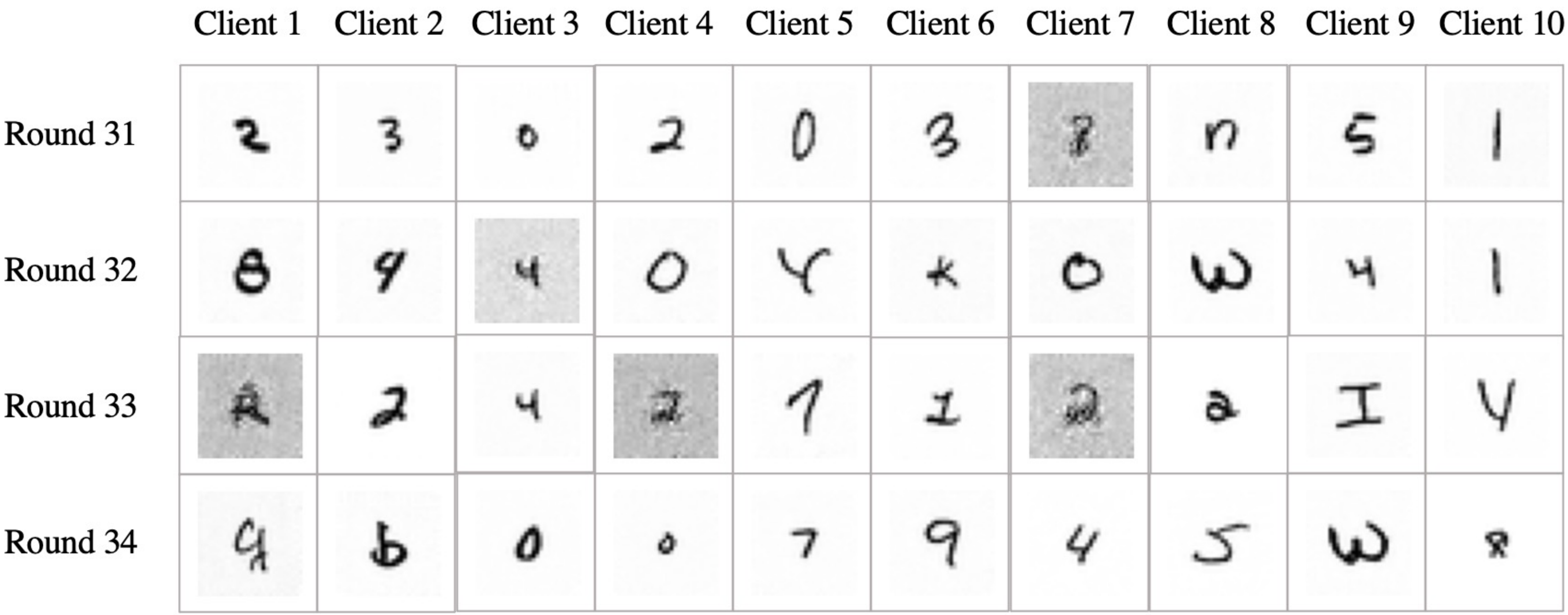
Maximum iteration

Learning rate of the optimization

- The command to run the example attack on FEMNIST with FedAvg:

```
python federatedscope/main.py --cfg
federatedscope/attack/example_attack_config/reconstruct_fedavg_
opt_on_femnist.yaml
```

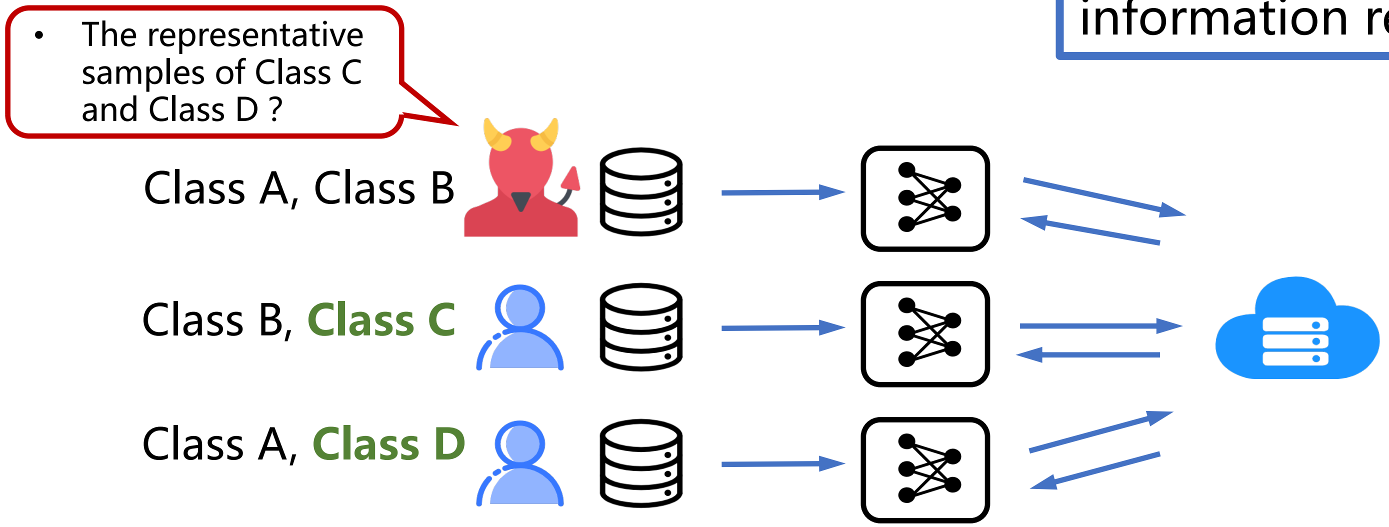
Training Data/Label Inference Attack



Class Representative Attack

- **Goal:** Infer representative samples of a specific class
- **Attacker Role:** Client

The client has data that only covers partial class labels, and is curious about information related to other classes!



Class Representative Attack

Active Attack

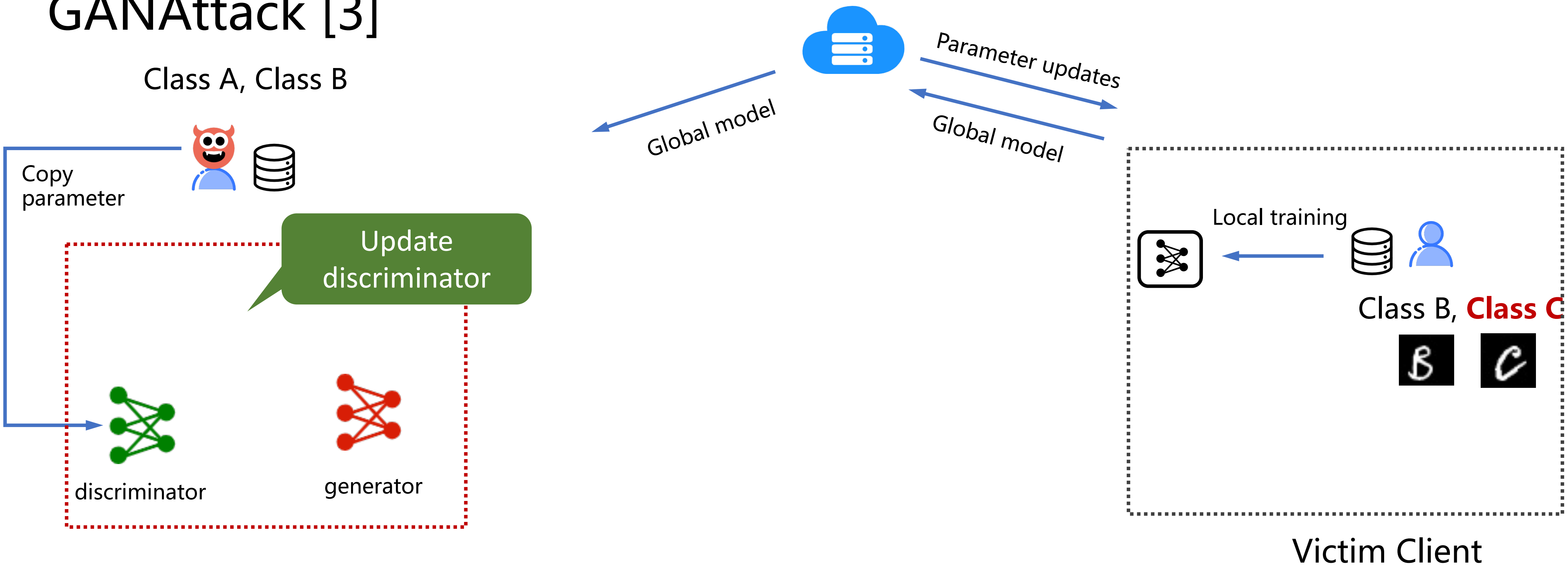
GANAttack [3]

- Attacker
  - Hold and update a local GAN:
    - Generate data that classified as target label by the global model
  - Inject the mislabeled data (with the target label):
    - Reveal more label related information by amplifying the impact of training data from other clients with the target label

[3] Deep models under the GAN: information leakage from collaborative deep learning. CCS. 2017.

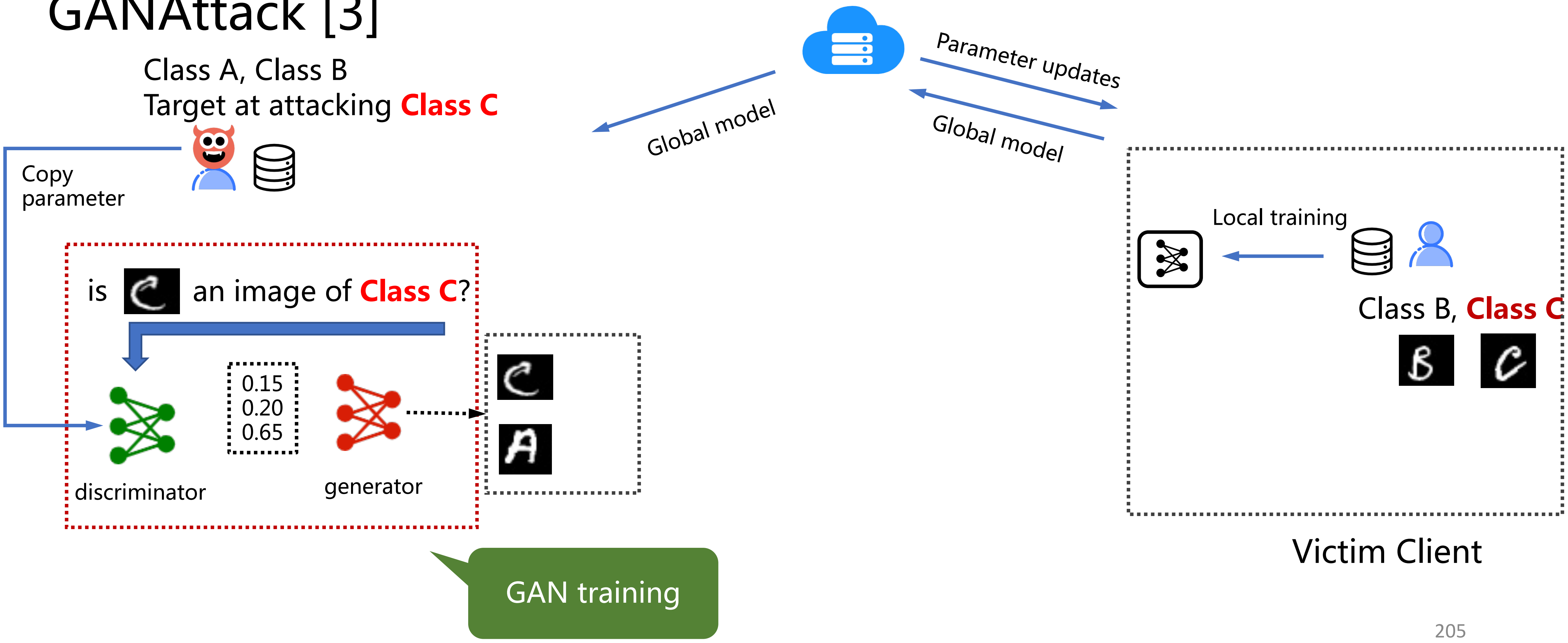
Class Representative Attack

GANAttack [3]



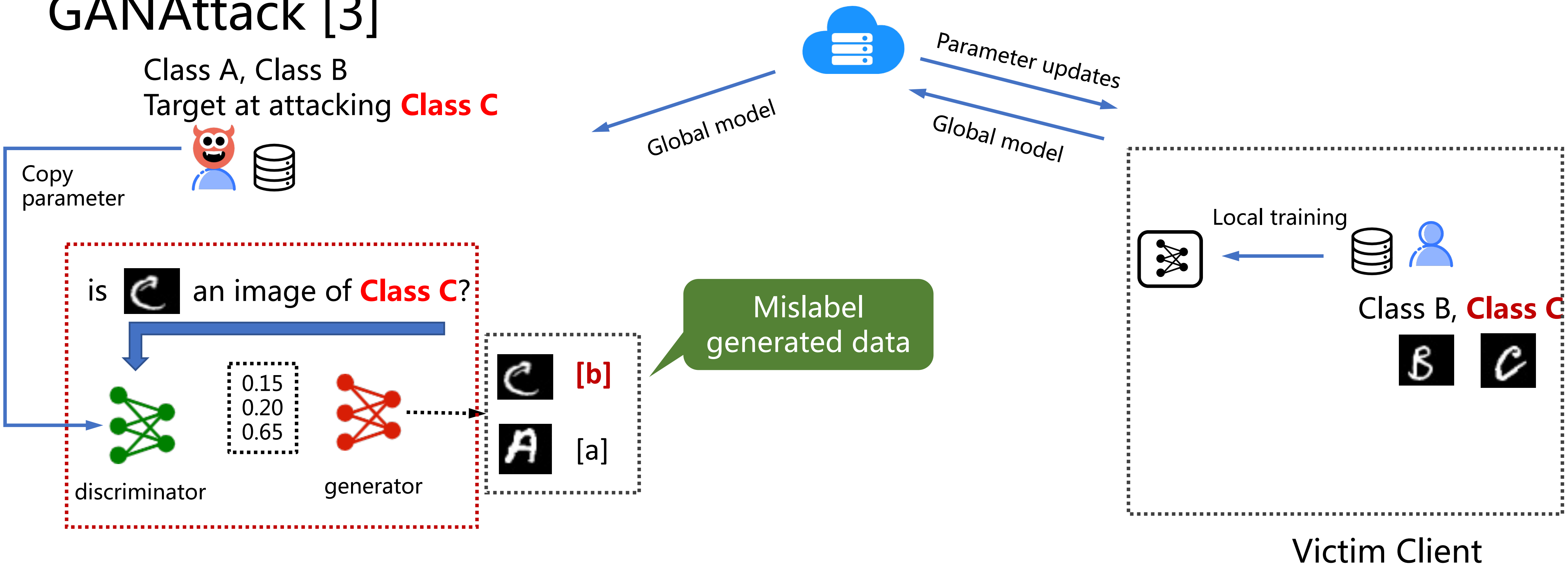
Class Representative Attack

GANAttack [3]



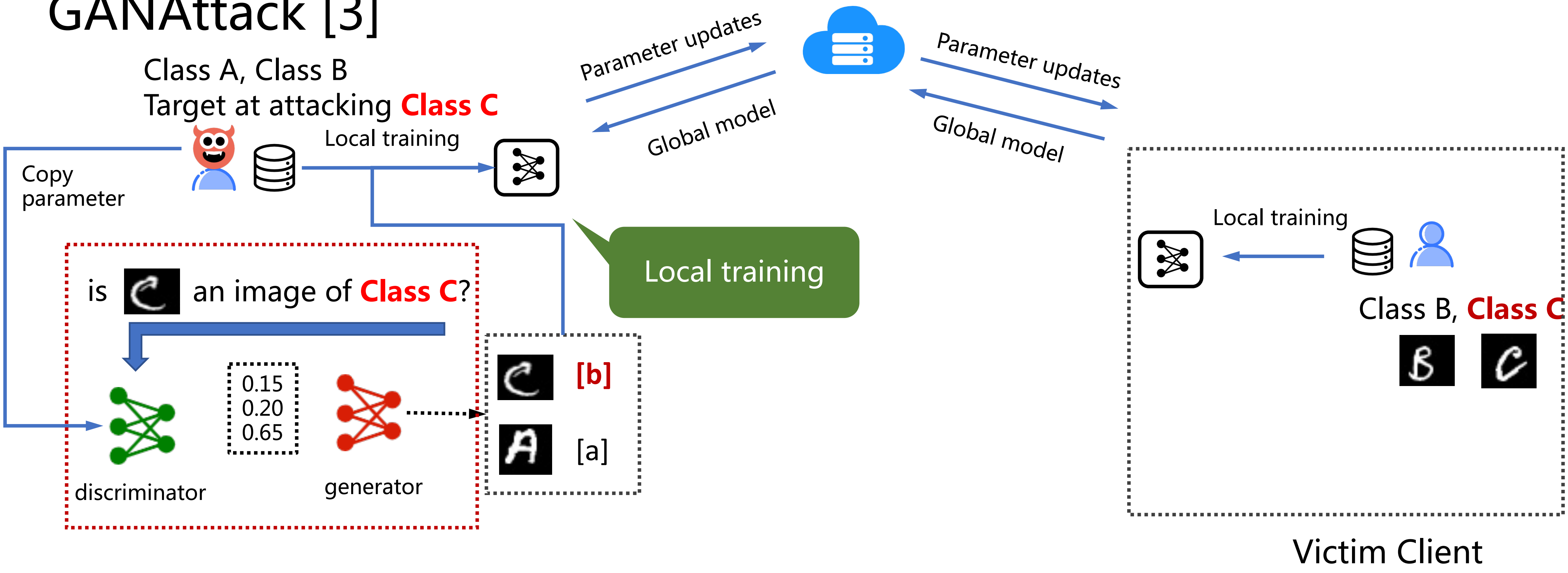
Class Representative Attack

GANAttack [3]



Class Representative Attack

GANAttack [3]



Class Representative Attack

Running GANAttack on FEMNIST in FederatedScope:

- Configuration (insert these lines into the current file):

```
attack:
  attack_method: gan_attack
  attacker_id: 5
  target_label_ind: 3
```

Malicious attacker id

The target class label index

- The command to run the example attack on FEMNIST with FedAvg:

```
python federatedscope/main.py --cfg
federatedscope/attack/example_attack_config/CRA_fedavg_convnet2
_on_femnist.yaml
```

Class Representative Attack

- Running Result:
  Images Generated by the generator:

Develop Attack Methods in FederatedScope

First make sure that the target FL algorithm has been implemented.

<u>Server as attacker:</u>

Attacks act during training:

Participant Plug-In

- Inherit the server class
- Add the attack actions by modifying `callback_funcs_model_para` function

Attacks act after training:

Participant Plug-In

- Inherit the server class
- Add the attack actions after the last round by modifying `callback_funcs_model_para` function

Example

Property Inference Attack

- Inherit `Server` class and modify `callback_funcs_model_para`

```
class PassivePIAServer(Server):
    ...
    def callback_funcs_model_para(self, message: Message):
        round, sender, content = message.state, message.sender, message.content
        # For a new round
```

Example

Property Inference Attack

- Inherit `Server` class and modify `callback_funcs_model_para`
  - At each FL round:
    - Collect the model updates
    - Generate training data for PIA classifier

```python
class PassivePIAServer(Server):
    ...
    def callback_funcs_model_para(self, message: Message):
        round, sender, content = message.state, message.sender, message.content
        # For a new round
        if round not in self.msg_buffer['train'].keys():
            self.msg_buffer['train'][round] = dict()

        self.msg_buffer['train'][round][sender] = content

        # collect the updates
        self.pia_attacker.collect_updates(previous_para= self.model.state_dict(),
                                          updated_parameter=content[1],
                                          round=round,
                                          client_id=sender)
        self.pia_attacker.get_data_for_dataset_prop_classifier(model=self.model)
```

Attacks act during training

Example

Property Inference Attack

- Inherit `Server` class and modify `callback_funcs_model_para`
    - At each FL round:
        - Collect the model updates
        - Generate training data for PIA classifier
    - After training:
        - Train PIA classifier
        - Inference the property

Attacks act after training

```python
class PassivePIAServer(Server):
...
def callback_funcs_model_para(self, message: Message):
        round, sender, content = message.state, message.sender, message.content
        # For a new round
        if round not in self.msg_buffer['train'].keys():
            self.msg_buffer['train'][round] = dict()

        self.msg_buffer['train'][round][sender] = content

        # collect the updates
        self.pia_attacker.collect_updates(previous_para= self.model.state_dict(),
                                          updated_parameter=content[1],
                                          round=round,
                                          client_id=sender)
        self.pia_attacker.get_data_for_dataset_prop_classifier(model=self.model)

        if self._cfg.federate.online_aggr:
            self.aggregator.inc(content)
        self.check_and_move_on()

        if self.state == self.total_round_num:
            self.pia_attacker.train_property_classifier()
            self.pia_results = self.pia_attacker.infer_collected()
            print(self.pia_results)
```

Check if it is the last round

Example

Property Inference Attack

- Inherit `Server` class and modify `callback_funcs_model_para`
  - At each FL round:
    - Collect the model updates
    - Generate training data for PIA classifier
  - After training:
    - Train PIA classifier
    - Inference the property

```python
class PassivePIAServer(Server):
```

...

```python
def callback_funcs_model_para(self, message: Message):
        round, sender, content = message.state, message.sender, message.content
        # For a new round
        if round not in self.msg_buffer['train'].keys():
            self.msg_buffer['train'][round] = dict()

        self.msg_buffer['train'][round][sender] = content

        # collect the updates
        self.pia_attacker.collect_updates(previous_para= self.model.state_dict(),
                                          updated_parameter=content[1],
                                          round=round,
                                          client_id=sender)
        self.pia_attacker.get_data_for_dataset_prop_classifier(model=self.model)

        if self._cfg.federate.online_aggr:
            self.aggregator.inc(content)
        self.check_and_move_on()

        if self.state == self.total_round_num:
            self.pia_attacker.train_property_classifier()
            self.pia_results = self.pia_attacker.infer_collected()
            print(self.pia_results)
```

Add the overloaded `Server` class to `get_server_cls` function

```python
def get_server_cls(cfg):
    if cfg.attack.attack_method.lower() in ['dlg', 'ig']:
        from federatedscope.attack.worker_as_attacker.server_attacker import PassiveServer
        return PassiveServer
    elif cfg.attack.attack_method.lower() in ['passivepia']:
        from federatedscope.attack.worker_as_attacker.server_attacker import PassivePIAServer
        return PassivePIAServer
```

Develop Attack Methods in FederatedScope

First make sure that the target FL algorithm has been implemented.

<u>Client as attacker:</u>

Behavior Plug-In

Attacks act during training:

- Wrap the `trainer` to add acttack actions

Participant Plug-In

Attacks act after training:

- Inherit the `Client` class
- Add the attack actions by modifying `callback_funcs_for_finish` function

Example

Class Representative Attack

- Wrap the `trainer`

```
def wrap_GANTrainer(
        base_trainer: Type[GeneralTrainer]) -> Type[GeneralTrainer]:
```

Attacks act during training

Example

Class Representative Attack

- Wrap the `trainer`
  - Hold a local GAN

```python
def wrap_GANTrainer(
        base_trainer: Type[GeneralTrainer]) -> Type[GeneralTrainer]:

    # ---------------- attribute-level plug-in -----------------------

    base_trainer.ctx.target_label_ind = base_trainer.cfg.attack.target_label_ind
    base_trainer.ctx.gan_cra = GANCRA(base_trainer.cfg.attack.target_label_ind,
                                      base_trainer.ctx.model,
                                      dataset_name = base_trainer.cfg.data.type,
                                      device=base_trainer.ctx.device,
                                      sav_pth=base_trainer.cfg.outdir
                                      )
```

Example

Class Representative Attack

- Wrap the `trainer`
  - Hold a local GAN
  - At each FL round:
    - Before local training:
      - Update GAN’ s discriminator by the received parameters, and train GAN’ s generator
      - Generate fake data and mislabel them

```python
def wrap_GANTrainer(
        base_trainer: Type[GeneralTrainer]) -> Type[GeneralTrainer]:

    # ---------------- attribute-level plug-in -----------------------

    base_trainer.ctx.target_label_ind = base_trainer.cfg.attack.target_label_ind
    base_trainer.ctx.gan_cra = GANCRA(base_trainer.cfg.attack.target_label_ind,
                                      base_trainer.ctx.model,
                                      dataset_name = base_trainer.cfg.data.type,
                                      device=base_trainer.ctx.device,
                                      sav_pth=base_trainer.cfg.outdir
                                      )

    # ---- action-level plug-in -------

    base_trainer.register_hook_in_train(new_hook=hook_on_gan_cra_train,
                                        trigger='on_batch_start',
                                        insert_mode=-1)
    base_trainer.register_hook_in_train(
        new_hook=hook_on_batch_injected_data_generation,
        trigger='on_batch_start',
        insert_mode=-1)
```

Example

Class Representative Attack

- Wrap the `trainer`
    - Hold a local GAN
    - At each FL round:
        - Before local training:
            - Update GAN’ s discriminator by the received parameters, and train GAN’ s generator
            - Generate fake data and mislabel them
        - During local bath forward:
            - Inject the fake data in training batch

```python
def wrap_GANTrainer(
        base_trainer: Type[GeneralTrainer]) -> Type[GeneralTrainer]:

    # ---------------- attribute-level plug-in -----------------------

    base_trainer.ctx.target_label_ind = base_trainer.cfg.attack.target_label_ind
    base_trainer.ctx.gan_cra = GANCRA(base_trainer.cfg.attack.target_label_ind,
                                      base_trainer.ctx.model,
                                      dataset_name = base_trainer.cfg.data.type,
                                      device=base_trainer.ctx.device,
                                      sav_pth=base_trainer.cfg.outdir
                                      )

    # ---- action-level plug-in -------

    base_trainer.register_hook_in_train(new_hook=hook_on_gan_cra_train,
                                        trigger='on_batch_start',
                                        insert_mode=-1)
    base_trainer.register_hook_in_train(
        new_hook=hook_on_batch_injected_data_generation,
        trigger='on_batch_start',
        insert_mode=-1)
    base_trainer.register_hook_in_train(
        new_hook=hook_on_batch_forward_injected_data,
        trigger='on_batch_forward',
        insert_mode=-1)

    return base_trainer
```

Defense Strategies

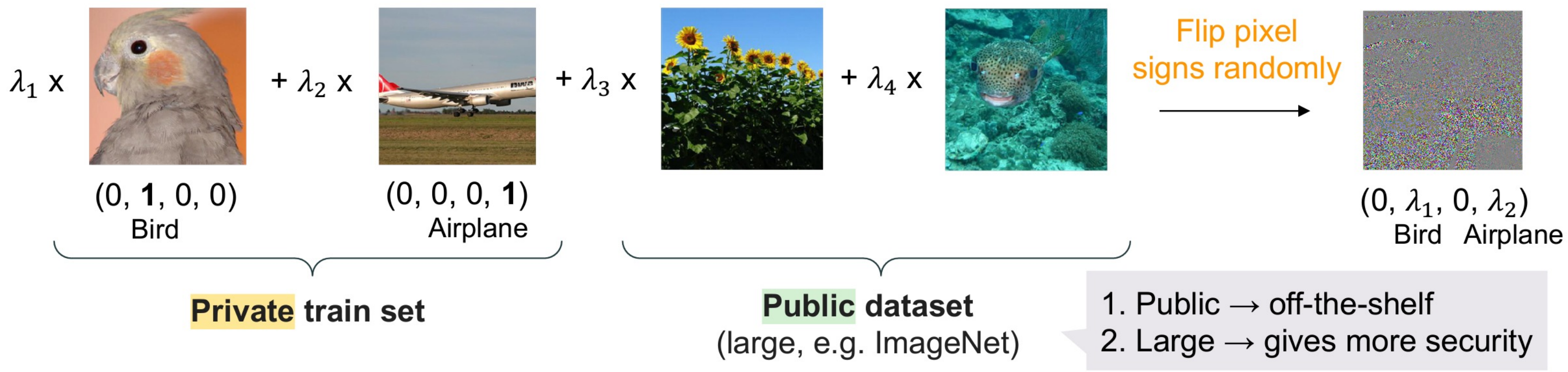
- Encrypt gradients
  - Secure aggregation, such as Multi-party computation (MPC);
  - Homomorphic encryption (HE)
- Perturbing gradients
  - Gradient pruning
  - Differential privacy: adding noise to gradient

FederatedScope supports the above defense strategies!

Defense Strategies

Wil be supported in FederatedScope soon!

- Weak encryption of inputs (i.e. encoding inputs)
  - MixUp [7]: create the images via linear combination of image pair
  - InstaHide[8]: extend MixUp



[7] Mixup: Beyond Empirical Risk Minimization. ICLR, 2018.
[8] InstaHide: Instance-hiding schemes for private distributed learning. ICML 2020.

Defense Strategies

- Add private components to the model (regularization, network):
    - Do not share BatchNorm layer during FL [9]
    - Secret Polarization Network [10]:
        - Some fully connected layers are kept private with its parameters not shared

Practical defense suggestions [9]:

- Use large batch size (>=32)
- Combine multiple defenses may achieve a better utility-privacy trade-off

[9] Evaluating Gradient Inversion Attacks and Defenses in Federated Learning. NeurIPS, 2021.

[10] Rethinking Privacy Preserving Deep Learning: How to Evaluate and Thwart Privacy Attacks. In *Federated Learning: Privacy and Incentive*. LNCS, 2020.

More Attack Methods in FederatedScope

- **Privacy attacks**
  - More SOTA privacy attacks
  - Defense strategies
- **Poisoning attacks**
  - Data poisoning
  - Model poisoning
  - Back-door
  - Defense strategies

References

[1] Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives. Cybersecur. 2022.

[2] Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. IEEE Symposium on Security and Privacy 2019.

[3] Deep models under the GAN: information leakage from collaborative deep learning. CCS. 2017.

[4] Deep leakage from gradients. NeurIPS 2019.

[5] idlg: Improved deep leakage from gradients[J]. arXiv preprint, 2020.

[6] Inverting gradients-how easy is it to break privacy in federated learning?. NeurIPS 2020.

[7] Mixup: Beyond Empirical Risk Minimization. ICLR, 2018.

[8] InstaHide: Instance-hiding schemes for private distributed learning. ICML 2020.

[9] Evaluating Gradient Inversion Attacks and Defenses in Federated Learning. NeurIPS, 2021.

[10] Rethinking Privacy Preserving Deep Learning: How to Evaluate and Thwart Privacy Attacks. In *Federated Learning: Privacy and Incentive*. LNCS, 2020.

Agenda of Tutorial

- Overview
- Personalized Federated Learning
- Federated Graph Learning
- Federated Hyperparameter Optimization
- Privacy Attacks

FederatedScope Team

Website: https://federatedscope.io/


Yaliang Li


Bolin Ding


Zhen Wang


Yuexiang Xie


Dawei Gao


Liuyi Yao


Daoyuan Chen


Weirui Kuang


Hongzhu Shi


Jingren Zhou

Thank you!

Yaliang Li, Zhen Wang, and Bolin Ding

Email: {yaliang.li, jones.wz, bolin.ding}@alibaba-inc.com

Please feel free to contact us if you have any questions,
or you are interested in **full-time** or **research intern** positions.