RAY

RAY FORWARD 2023

下一代 AI 计算

07/02
BEIJING

Add：北京市朝阳区东三环环球金融中心（WFC）东塔 9F

RAY

Ray 在隐语联邦学习中的实践

隐语联邦学习工程师 胡东文

Effortlessly scale your most complex workloadsz

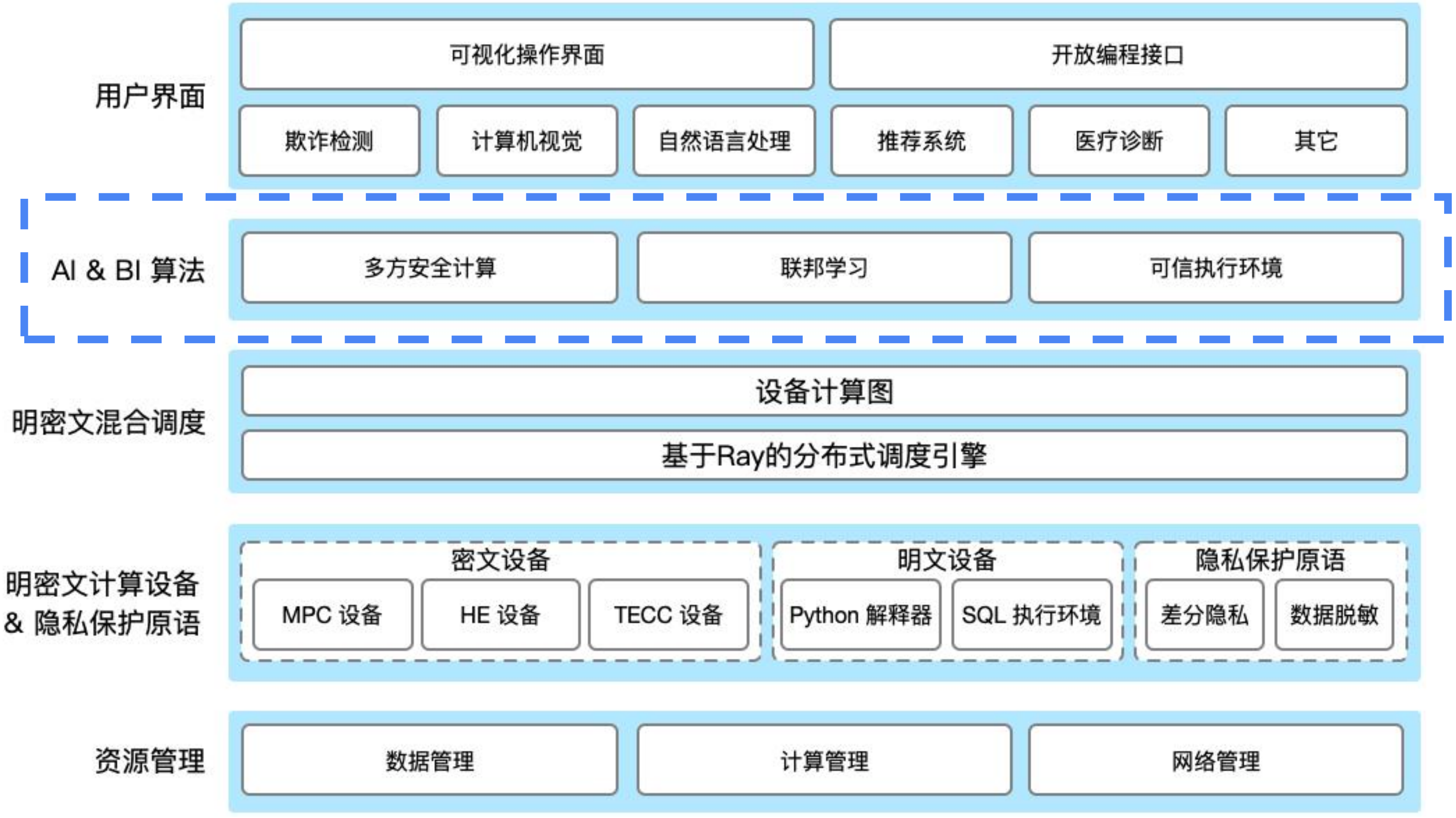
RAY

胡东文

蚂蚁集团-隐私计算部-隐语 联邦学习工程师

Effortlessly scale your most complex workloadsz

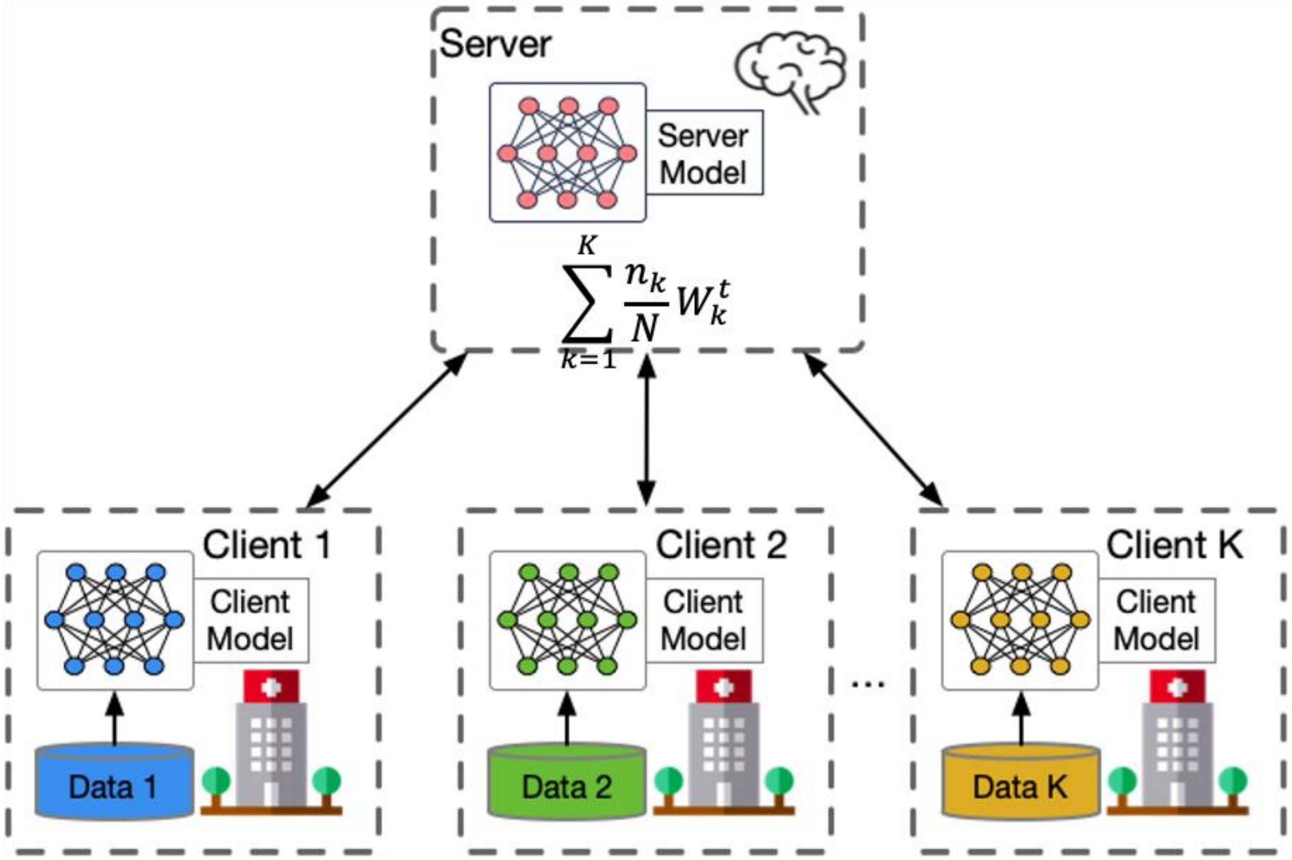
隐语 SecretFlow



SecretFlow 是一个隐私保护数据分析和机器学习的统一框架。

- **完备性**：支持多种隐私计算技术，可灵活组装，满足不同场景需求。
- **透明性**：构建统一的技术框架，尽量让底层技术迭代对上层透明应用，具有高内聚和低耦合。
- **开放性**：不同专业方向的人可以轻松参与框架的建设，共同加速隐私计算技术的发展。
- **连接性**：不同底层技术支持的场景中的数据可以相互连接。

联邦学习的编程“难题”

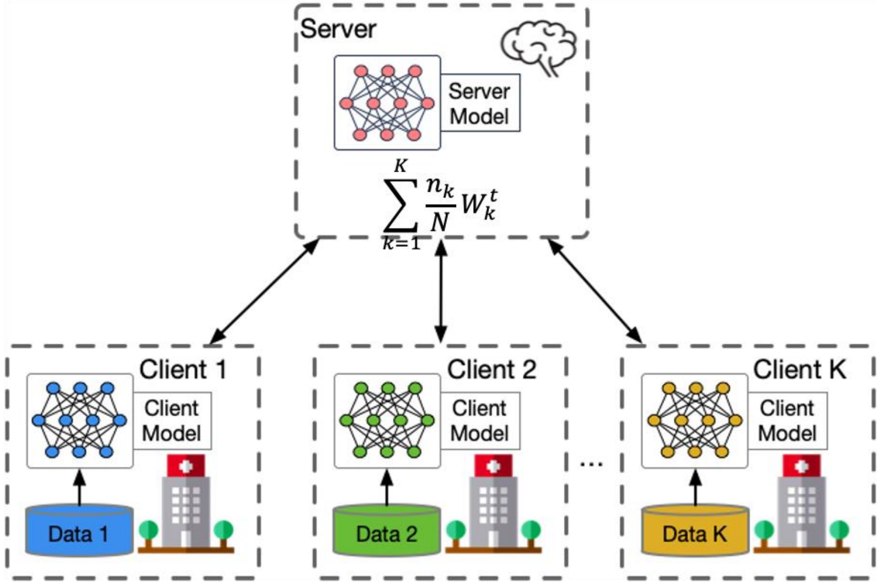


- 隐私计算类型跨机构分布式程序的难点
  1. 没有可信的调度中心(Driver)
  2. 所有数据传递都需要明确且可控

联邦学习的编程“难题”

```python
# server
weights = init_model_weights()
for c in clients:
  send(c, weights)


for _ in range(epochs):
  all_weights = []
  for c in clients:
    all_weights.append(recv(c))

  weights = average(all_weights)
  for c in clients:
    send(c, weights)
```

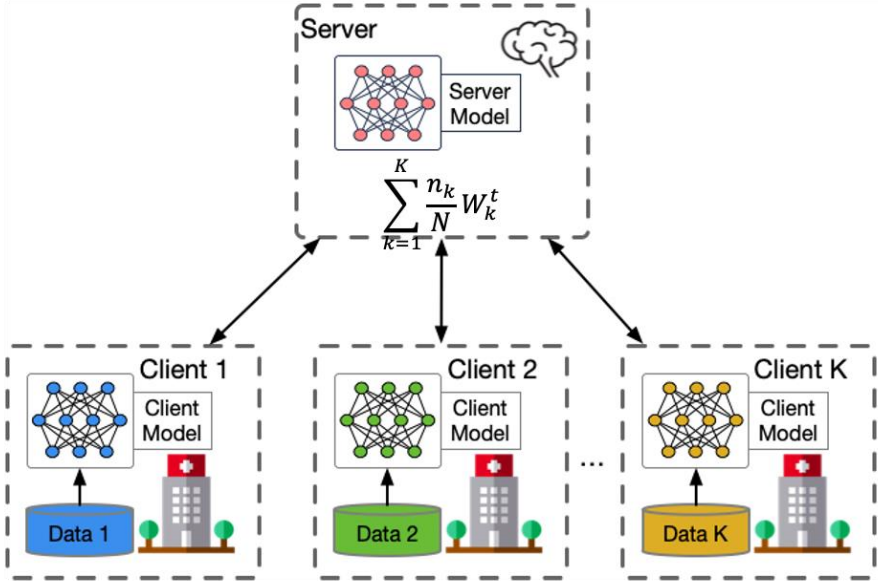
```python
# client


weights = recv(server)
model.set_weights(weights)

for _ in range(epochs):
  local_train(model)
  weights = model.get_weights()
  send(server, weights)



  weights = recv(server)
  model.set_weights(weights)

model.save()
```



联邦学习的编程“难题”

```python
# server
@register(INIT)
def init_handler():
  weights = init_model_weights()
  triger(UPDATE_WEIGHTS, weights)


@register(AVERAGE)
def average_handler(weights):
  global all_weights
  all_weights.append(weights)
  if len(all_weights) == client_num:
    return
  weights = average(all_weights)
  triger(UPDATE_WEIGHTS, weights)
```

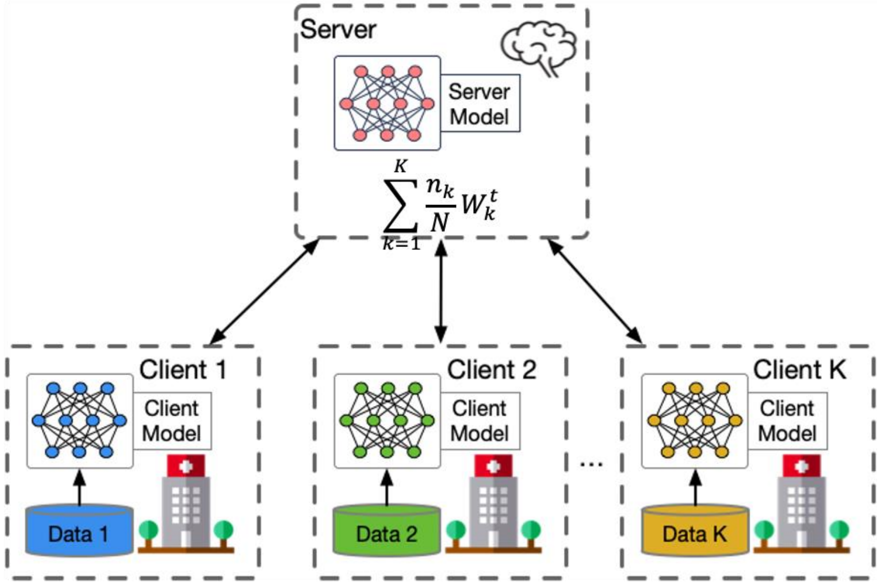
```python
# client
@register(UPDATE_WEIGHTS)
def local_train_handler(weights):
  global epoch, epochs
  model.set_weights(weights)
  local_train(model)
  epoch +=1
  if epoch < epochs:
    weights = model.get_weights()
    triger(AVERAGE, weights)
  else:
    model.save()
    triger(END)
```



Ray 与隐语的中心化视角编程

```python
import ray

@ray.remote
class Client:
  def set_weights(self, weights):
    ...

  def get_weights(self):
    ...

  def local_train(self):
    ...

  def save_model(self):
    ...

@ray.remote
def init_weights():
  ...

@ray.remote
def average(all_weights):
  ...
```

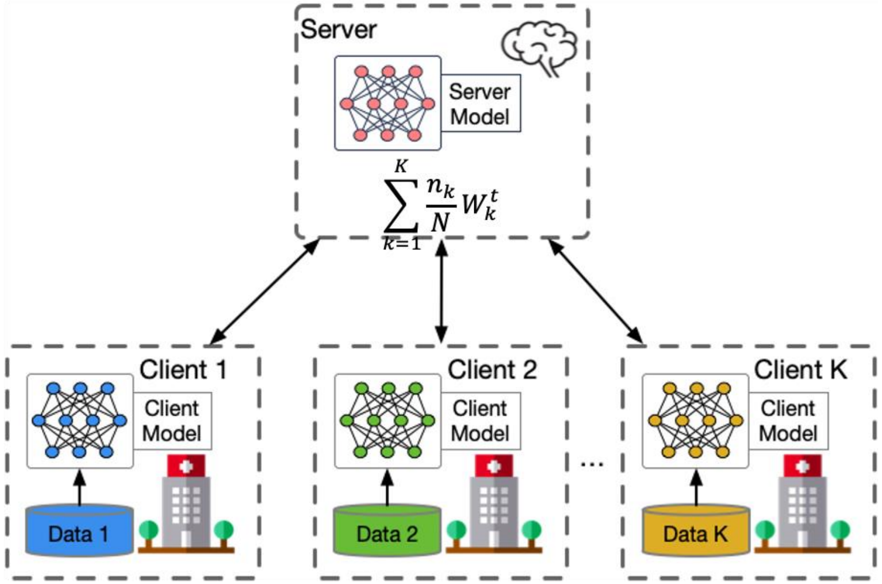
```python
# server = ?
clients = [Client.remote() for
           _ in range(client_num)]
# 初始化参数
weights = init_weights.remote()
for _ in range(epochs):
  all_weights = []
  for c in clients:
    # client 端更新参数并进行本地训练
    c.set_weights.remote(weights)
    c.local_train.remote()
    new_weights = c.get_weights.remote()
    # 收集更新后的参数列表
    all_weights.append(new_weights)
  # 在 server 端做参数聚合
  weights = average.remote(all_weights)

for c in clients:
  # 训练结束，保存模型
  c.save_model.remote()
```



Ray 与隐语的中心化视角编程

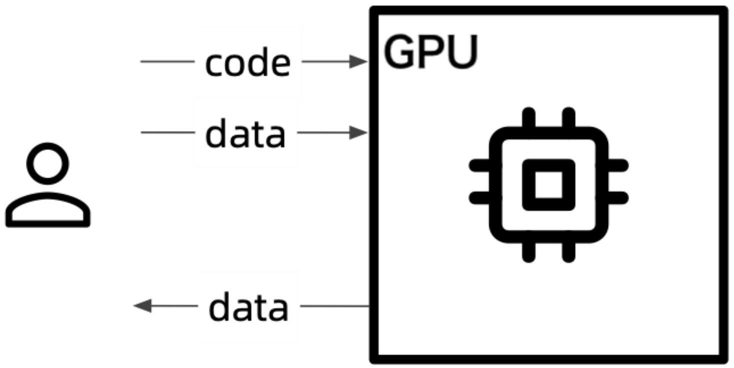
```python
import secretflow as sf

@sf.proxy(sf.PYUObject)
class Client:
  def set_weights(self, weights):
    ...

  def get_weights(self):
    ...

  def local_train(self):
    ...

  def save_model(self):
    ...

# server
def init_weights():
  ...

def average(all_weights):
  ...
```

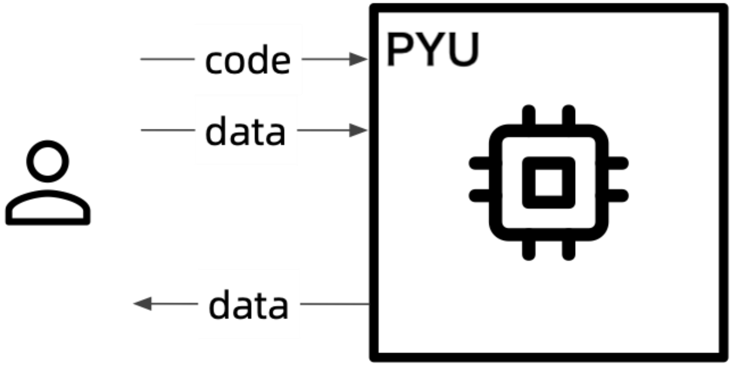
```python
server = sf.PYU(server_id)
clients = [Client(device=sf.PYU(client_id))
           for client_id in client_ids]
# 初始化参数
weights = server(init_weights)()
for _ in range(epochs):
  all_weights = []
  for c in clients:
    # client 端更新参数并进行本地训练
    weights = weights.to(c)
    c.set_weights(weights)
    c.local_train()
    weights_client = c.get_weights()
    # 将更新后的参数传到 server
    weights_server = weights_client.to(server)
    all_weights.append(weights_server)
  # 在 server 端做参数聚合
  weights = server(average)(all_weights)

for c in clients:
  # 训练结束，保存模型
  c.save_model()
```



https://github.com/ray-project/rayfed

在 Ray/RayFed 上构建设备抽象

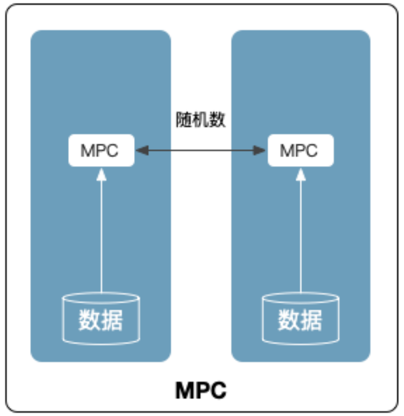


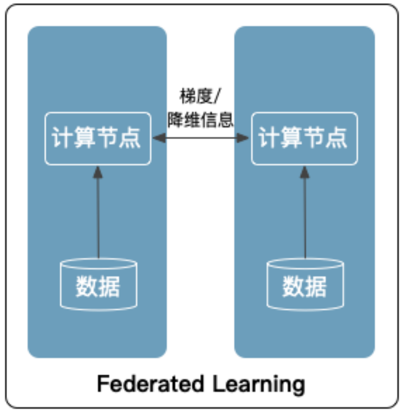


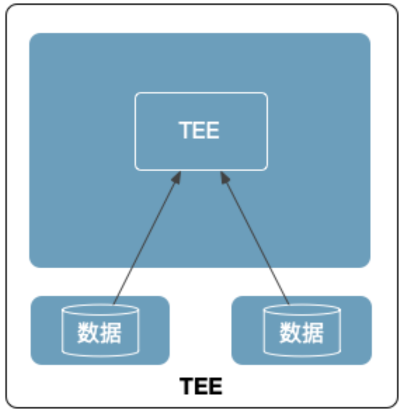

```python
import torch

gpu0 = torch.device('cuda:0')
x_cpu = torch.tensor([1,2,3])
x_gpu0 = x_cpu.to(gpu0)
```

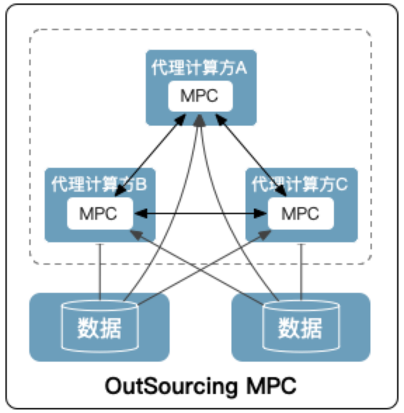
```python
import secretflow as sf

server = sf.PYU(‘server')
...
weights_server = weights_client.to(server)
all_weights.append(weights_server)
...
weights = server(average)(all_weights)
```

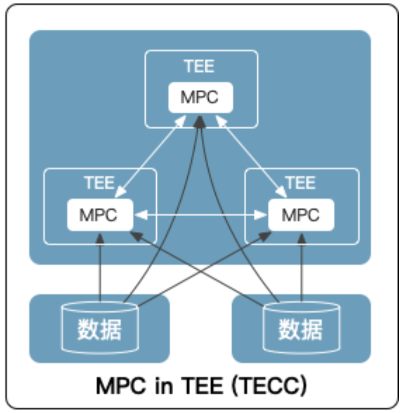
在 Ray/RayFed 上构建设备抽象











差分隐私

同态加密

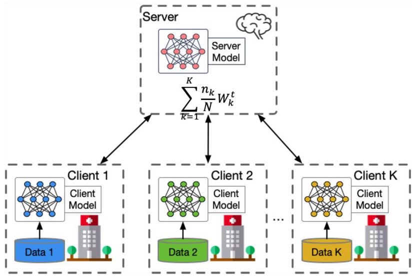
隐私计算技术路线繁多，架构各异。

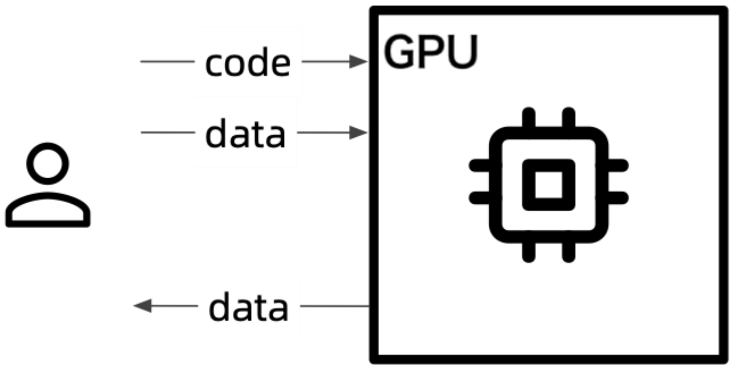
联邦学习实际上是一个明密文混合的分布式机器学习范式。

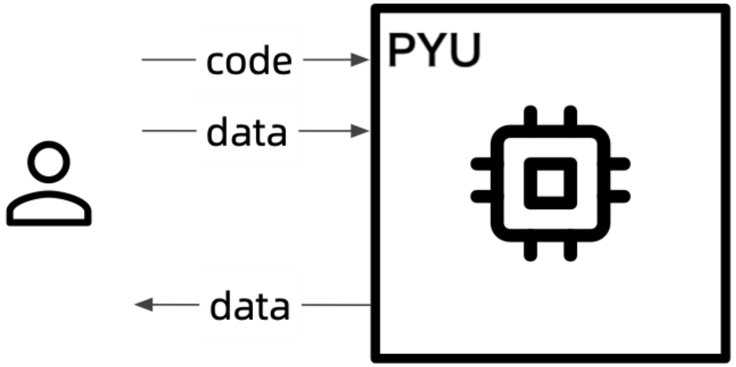
虚拟设备将多种隐私计算技术有机融合，形成一套灵活的编程框架

RAY

在 Ray/RayFed 上构建设备抽象



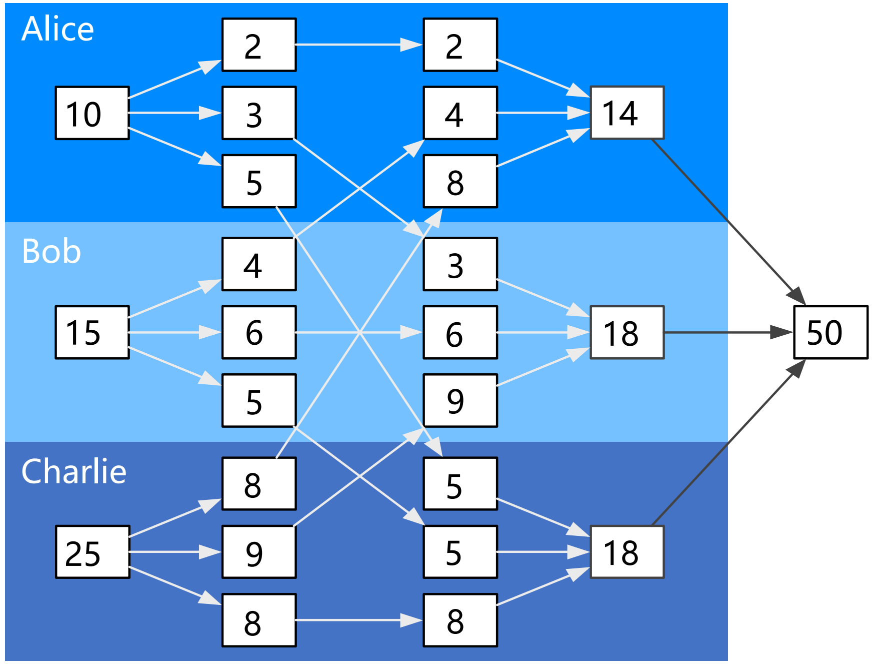





```
import torch

gpu0 = torch.device('cuda:0')
x_cpu = torch.tensor([1,2,3])
x_gpu0 = x_cpu.to(gpu0)
```

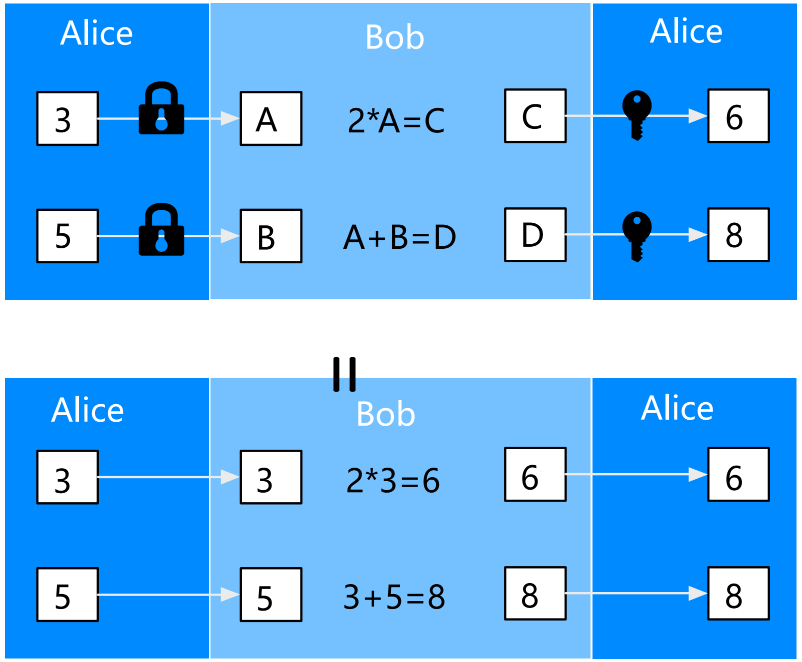
```
import secretflow as sf

server = sf.PYU('server')
...
weights_server = weights_client.to(server)
all_weights.append(weights_server)
...
weights = server(average)(all_weights)
```

在 Ray/RayFed 上构建设备抽象



秘密分享 - Secret Sharing （SS）

通过交换随机数完成协同计算

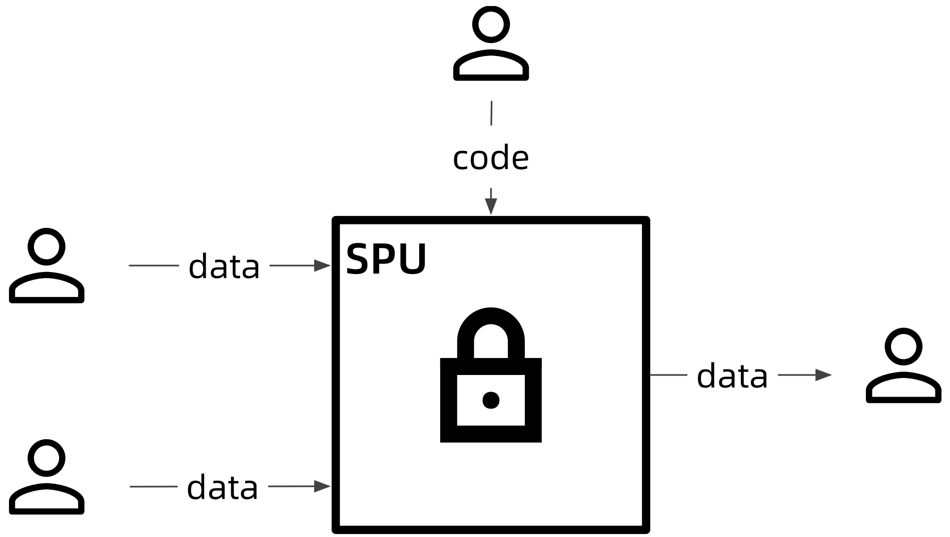


同态加密 - Homomorphic Encryption（HE）

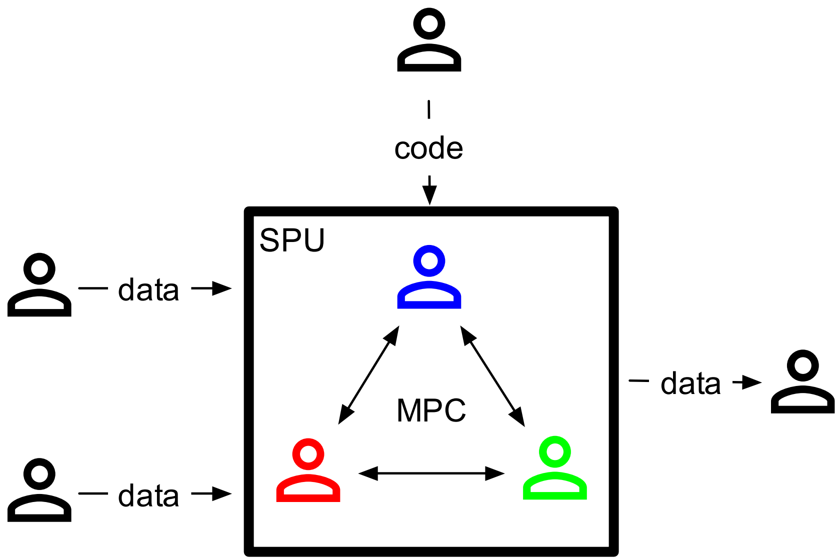
通过对加密数据进行运算再解密以达到跟直接对原数据进行运算结果相同的目的

RAY

在 Ray/RayFed 上构建设备抽象



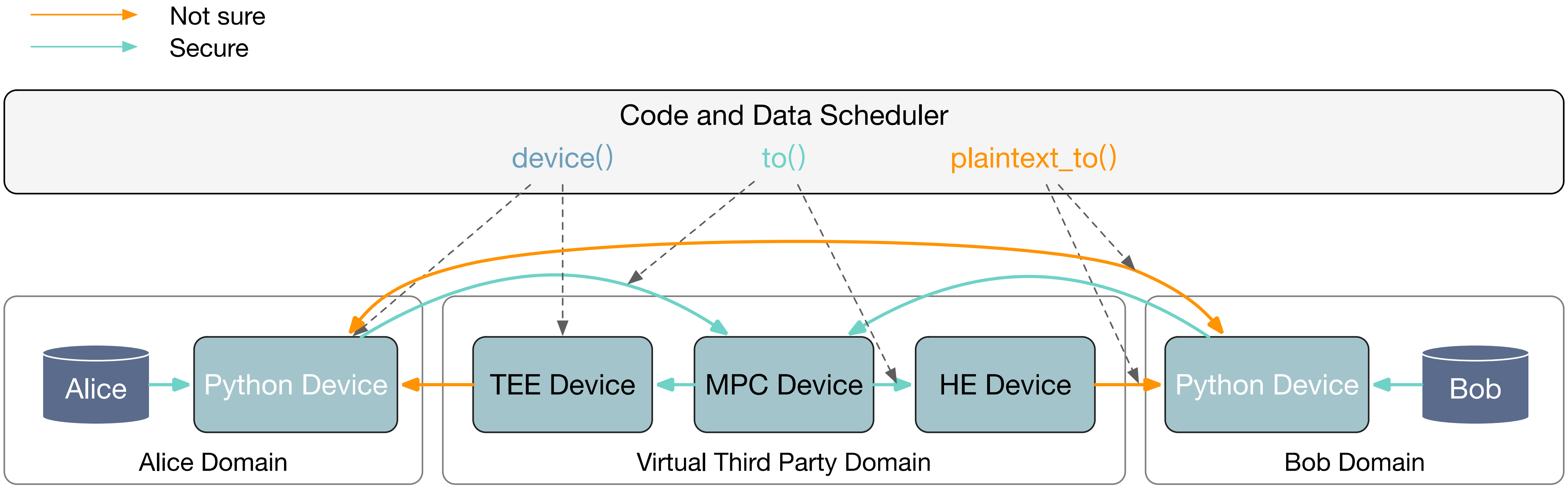
SPU 虚拟设备



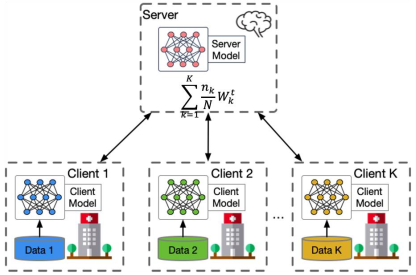
虚拟设备的物理组成

在 Ray/RayFed 上构建设备抽象

Computation = Device + Data Flow → SecretFlow



在 Ray/RayFed 上构建设备抽象




```python
import jax.numpy as jnp
import secretflow as sf


@sf.proxy(sf.PYUObject)
class Client:
  ...


# server
def average(all_weights):
  result = []
  for elements in zip(*all_weights):
    avg = jnp.average(
          jnp.array(elements),
          axis=0)
    result.append(avg)
  return result
```

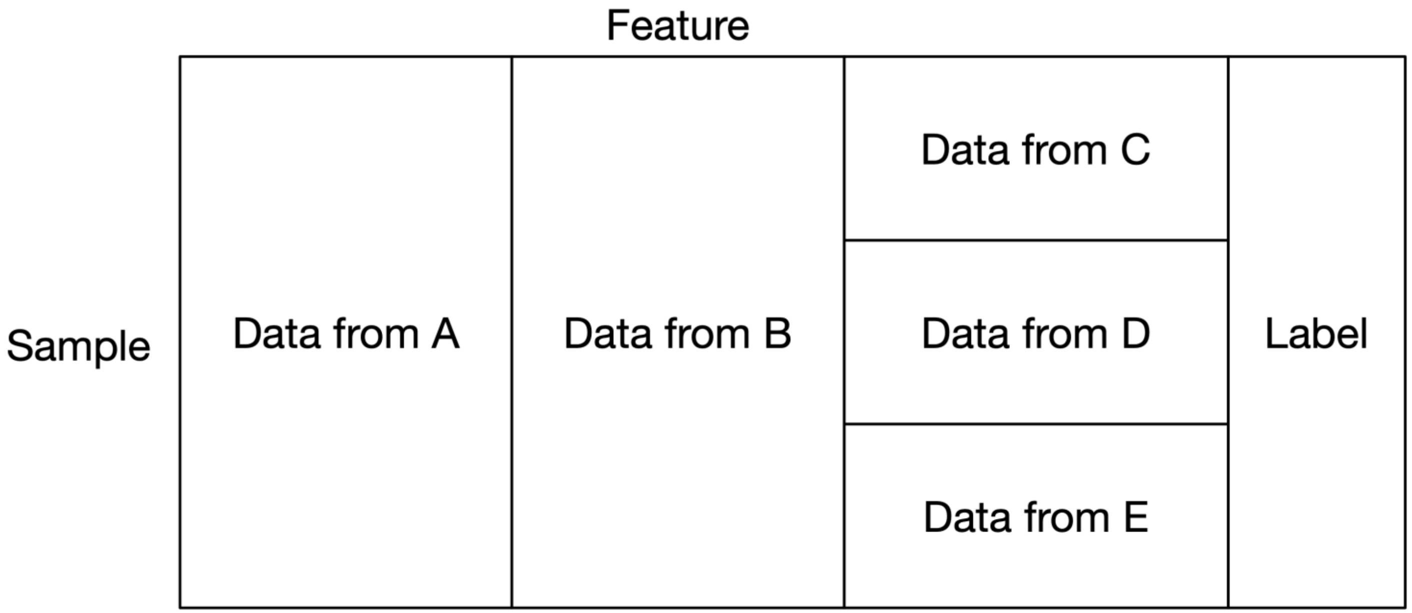
使用 PYU 聚合

```python
server = sf.PYU(server_id)
clients = [Client(device=sf.PYU(client_id))
           for client_id in client_ids]
# 初始化参数
weights = server(init_weights)()
for _ in range(epochs):
  all_weights = []
  for c in clients:
    # client 端更新参数并进行本地训练
    weights = weights.to(c)
    c.set_weights(weights)
    c.local_train()
    weights_client = c.get_weights()
    # 将更新后的参数传到 server
    weights_server = weights_client.to(server)
    all_weights.append(weights_server)
  # 在 server 端做参数聚合
  weights = server(average)(all_weights)

for c in clients:
  # 训练结束，保存模型
  c.save_model()
```

使用 SPU 聚合

```python
server = sf.SPU(...) # 多方组成一个 SPU
clients = [Client(device=sf.PYU(client_id))
           for client_id in client_ids]
# 初始化参数
weights = server(init_weights)()
for _ in range(epochs):
  all_weights = []
  for c in clients:
    # client 端更新参数并进行本地训练
    weights = weights.to(c)
    c.set_weights(weights)
    c.local_train()
    weights_client = c.get_weights()
    # 将更新后的参数传到 server
    weights_server = weights_client.to(server)
    all_weights.append(weights_server)
  # 在 server 端做参数聚合
  weights = server(average)(all_weights)

for c in clients:
  # 训练结束，保存模型
  c.save_model()
```

复杂一点的例子：混合 LR



A和B拥有相同的样本但是不同的特征；
C、D、E拥有不同的样本但是特征相同；

复杂一点的例子：混合 LR

垂直联邦 LR

Vertical data 0	Data0 from A	Data0 from B	Data from C	Label0
Vertical data 1	Data1 from A	Data1 from B	Data from D	Label1
Vertical data 2	Data2 from A	Data2 from B	Data from E	Label2

水平联邦 LR

算法概览：
1.对垂直数据的多个数据分块进行垂直联邦逻辑回归。
2.对多个垂直数据进行水平联邦逻辑回归。

RAY

复杂一点的例子：混合 LR

Vertical data 0	Data0 from A	Data0 from B	Data from C	Label0
Vertical data 1	Data1 from A	Data1 from B	Data from D	Label1
Vertical data 2	Data2 from A	Data2 from B	Data from E	Label2

```python
class FlLogisticRegressionMix:
  def _agg_weights(self):
    weights_list = self.ver_lr_list.get_weight()
    agg_weight = [
      self.aggregators[i].average(weights, axis=0)
      for i, weights in enumerate(zip(*weights_list))
    ]
    self.ver_lr_list.set_weight(agg_weight)

  def fit(self, x, y, batch_size, epochs):
    for epoch in range(epochs):
      self._fit_in_steps(x, batch_size, epoch)
      self._agg_weights()

  def _fit_in_steps(self, x, batch_size, epoch):
    for ver_lr, x_part in zip(self.ver_lr_list, x.partitions):
      n_step = math.ceil(x_part.shape[0] / batch_size)
      ver_lr.fit_in_steps(n_step, epoch)
```

```python
class FlLogisticRegressionVertical:

  def fit_in_steps(self, n_step: int):
    for _ in range(n_step):
      # step 1: y device compute residual, send to current
      # device in HE ciphertext.
      x_batchs, y_batch = self._next_batch()
      h = self.predict(x_batchs)
      r = self.workers[self.y_device].compute_residual(y_batch, h)
      self.workers[self.y_device].update_weight_agg(
        x_batchs[list(self.workers.keys()).index(self.y_device)],r)
    for i, (device, worker) in enumerate(self.workers.items()):
      if device == self.y_device:
        continue
      # step 2: current device compute the gradients locally,
      # and add mask and send to y device.
      x_heu = worker.encode(x_batchs[i]).to(self.heu)
      r_heu = r.to(self.heu)
      m_heu = worker.generate_rand_mask().to(self.heu)
      maskg_heu = r_heu @ x_heu + m_heu
      # step 3: y device decrypts the masked gradients and send
      # back to current device.
      maskg = (
        self.workers[self.y_device]
        .decode(maskg_heu.to(self.y_device), self.fxp_bits)
        .to(device)
      )
      # step 4: update Wi.
      worker.update_weight(maskg)
```

SECRETFLOW 隐语

https://github.com/secretflow

https://gitee.com/secretflow

期待您的使用和共建

RAY

Thank you

C R E A T I V E   P O W E R P O I N T   T E M P L A T E

Effortlessly scale your most complex workloadsz