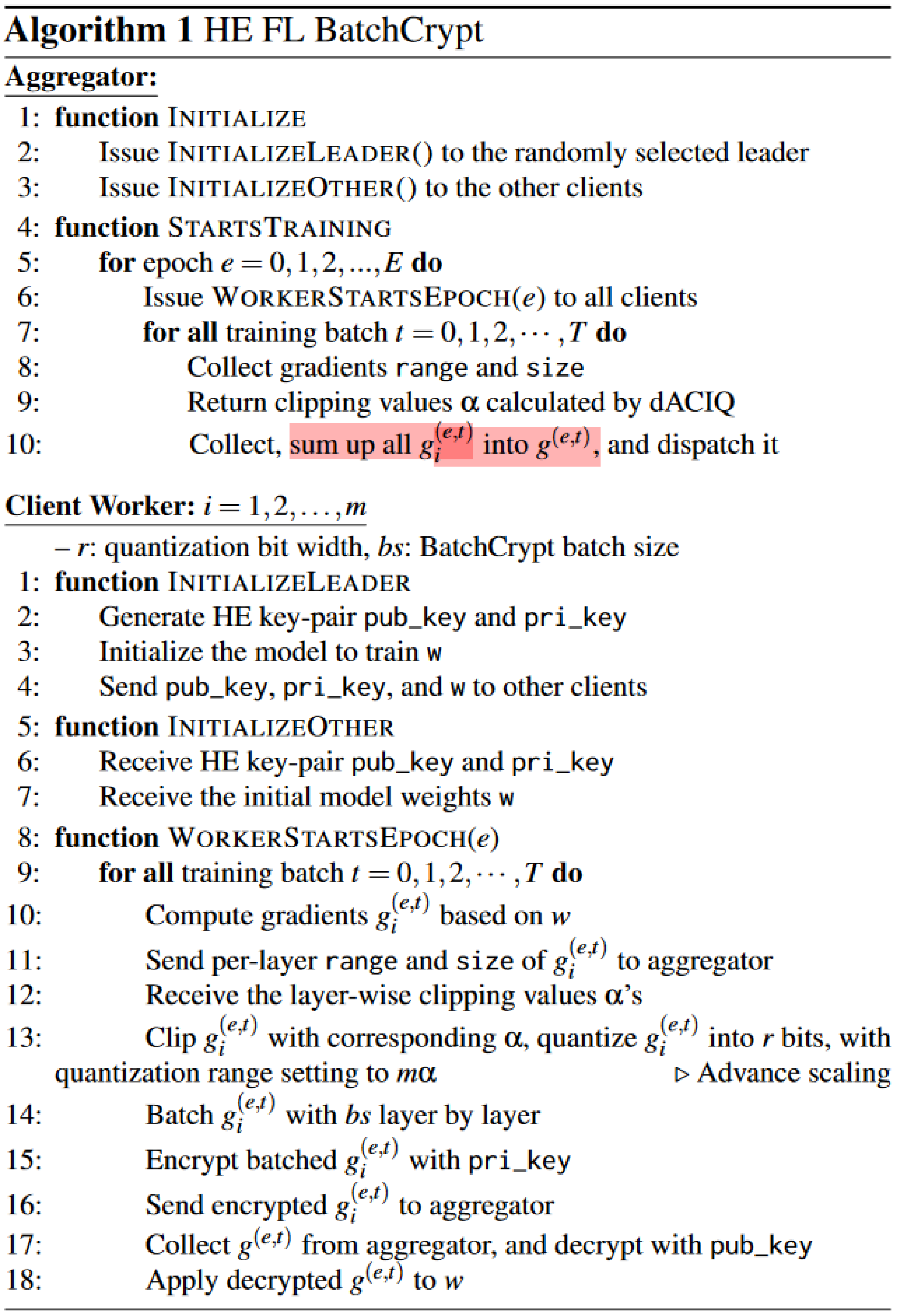
黄色的两篇感觉是比较solid的)

FL+PHE加速

- BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning 2020 USENIX
  - 算法过程:



Algorithm 1 HE FL BatchCrypt

```
Aggregator:
 1: function INITIALIZE
 2:     Issue INITIALIZELEADER() to the randomly selected leader
 3:     Issue INITIALIZEOTHER() to the other clients
 4: function STARTSTRAINING
 5:     for epoch $e = 0, 1, 2, ..., E$ do
 6:         Issue WORKERSTARTSEPOCH($e$) to all clients
 7:         for all training batch $t = 0, 1, 2, \cdots, T$ do
 8:             Collect gradients range and size
 9:             Return clipping values $\alpha$ calculated by dACIQ
10:             Collect, sum up all $g_i^{(e,t)}$ into $g^{(e,t)}$, and dispatch it

Client Worker: $i = 1, 2, \ldots, m$
    – $r$: quantization bit width, $bs$: BatchCrypt batch size
 1: function INITIALIZELEADER
 2:     Generate HE key-pair pub_key and pri_key
 3:     Initialize the model to train w
 4:     Send pub_key, pri_key, and w to other clients
 5: function INITIALIZEOTHER
 6:     Receive HE key-pair pub_key and pri_key
 7:     Receive the initial model weights w
 8: function WORKERSTARTSEPOCH($e$)
 9:     for all training batch $t = 0, 1, 2, \cdots, T$ do
10:         Compute gradients $g_i^{(e,t)}$ based on $w$
11:         Send per-layer range and size of $g_i^{(e,t)}$ to aggregator
12:         Receive the layer-wise clipping values $\alpha$'s
13:         Clip $g_i^{(e,t)}$ with corresponding $\alpha$, quantize $g_i^{(e,t)}$ into $r$ bits, with
            quantization range setting to $m\alpha$                ▷ Advance scaling
14:         Batch $g_i^{(e,t)}$ with $bs$ layer by layer
15:         Encrypt batched $g_i^{(e,t)}$ with pri_key
16:         Send encrypted $g_i^{(e,t)}$ to aggregator
17:         Collect $g^{(e,t)}$ from aggregator, and decrypt with pub_key
18:         Apply decrypted $g^{(e,t)}$ to $w$
```

所有的clients同一个HE key-pair

  - gradient clip:

分层进行的原因： gradients from different layers have different distributions

梯度需要随机的四舍五入stochastic rounding

如何确定clipping threshold α：

先前的研究表明，来自同一层的梯度分布是一种接近高斯分布的钟形bell-shaped分布。采用Scalable

methods for 8-bit training of neural networks. In *NeurIPS* (2018).中的技术进行**高斯拟合。**

- dACIQ：改变最先进的剪切技术ACIQ以适应BatchCrypt的对称量化+明文下执行distribution fifitting

  假设梯度遵循高斯分布X~N（0，σ2）。设q_i是第i个量化水平。我们计算Batchrypt中的累积误差如下：

$$E[(X-Q(X))^2]=\int_{-\infty}^{-\alpha} f(x)\cdot(x+\alpha)^2dx+\int_{\alpha}^{\infty} f(x)\cdot(x-\alpha)^2dx$$

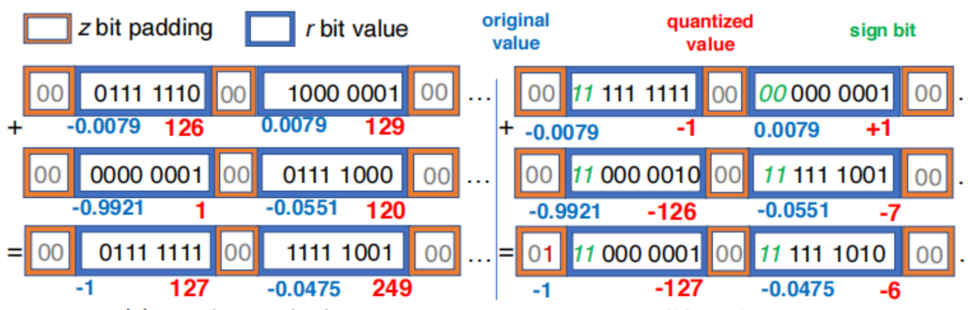
$$+\sum_{i=0}^{2^r-3}\int_{q_i}^{q_{i+1}} f(x)\ \cdot[\ (x-q_i)^2\cdot(\frac{q_{i+1}-x}{\triangle})\ +\ (x-q_{i+1})^2\cdot(\frac{x-q_i}{\triangle})\ ]dx$$

$$\approx\frac{\alpha^2+\sigma^2}{2}\cdot[1-erf(\frac{\alpha}{\sqrt{2}\sigma})]-\frac{\alpha\cdot\sigma\cdot e^{-\frac{\alpha^2}{2\cdot\sigma^2}}}{\sqrt{2\pi}}+\frac{2\alpha^2\cdot(2^r-2)}{3\cdot 2^{3r}},$$

  其中第一项和第二项是clip noise，第三项是舍入噪声rounding noise。*r*：r-bit quantization width，**只要我们知道σ，我们就可以从等式中得到最优阈值α**

 为了防止梯度加和时溢出，将量化范围设置为clip范围的m倍

- 量化：



(a) generic quantization (b) BatchCrypt

  - generic quantization:

    假设梯度 $g\in[-1,1]$，现在要将其量化为8-bit 无符号整数 (unsigned integer)。量化函数 $Q(\cdot)$ 和解量化函数 $Q^{-1}(\cdot)$ 定义如下:

    $Q(g)=\left[255\cdot\frac{g-\min}{\max-\min}\right]$，其中 [] 是就近取整函数 (rounding function)；

    $Q^{-1}\left(q_n\right)=q_n\cdot\frac{\max-\min}{255}+n\cdot\min$，其中 q_n 是 n 个量化梯度之和。

    但是，上述量化方法存在如下的问题:

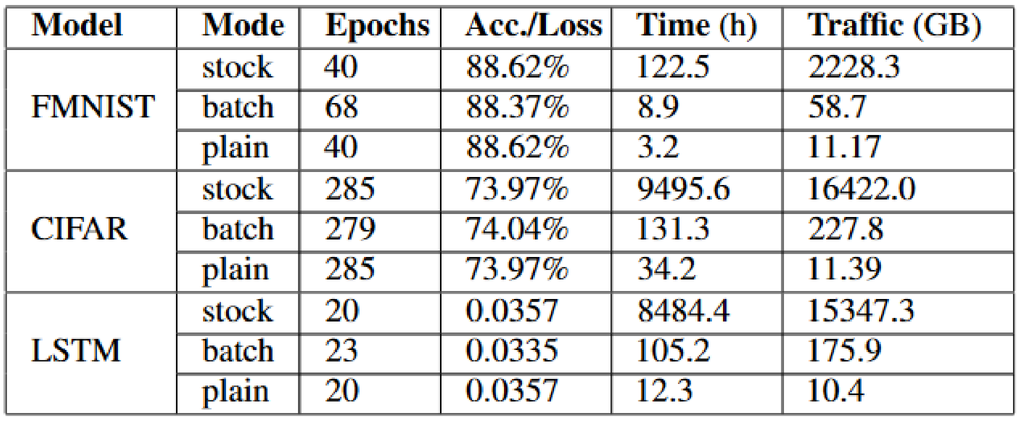
1. 要正确计算 $Q^{-1}(\cdot)$，必须事先预知 n。因此，每次加入新的用户，需要手动检验调整 n 的取值;
2. 容易溢出。因为所有的正负梯度都编码为无符号整数，多个用户的累加值容易导致溢出;
3. 不能区分正溢出和负溢出

  - BatchCrypt:

1. 有符号量化: 梯度被量化为有符号的整数，这样一来正负值相互抵消有助于解决溢出问题;

2. 关于原点对称的量化区间: 为了保证相反符号数能够正确的抵消，量化区间必须关于原点对称。 否则，假设将 $[-1,1]$ 量化到 $[-128,127]$ ，那么 $(-1+1) \Rightarrow (-128+127) = -1$ ，结 果错误;
3. 均匀量化。这是为了保证量化数值的加同态性质需要满足的性质。
   具体方法: Zhang等人针对梯度 $[-\alpha,\alpha]$ 提出了如下的量化方法，粗略来说便是将 $[-\alpha,0]$ 量化 到 $[-(2^r-1),0]$ ，将 $[0,\alpha]$ 量化到 $[0,(2^r-1)]$ 。公式化表示如下:
   $$Q(g)=\begin{cases}\lceil g\cdot 2^r\rceil, g\in[-\alpha,0];\\ \lfloor g\cdot 2^r\rfloor, g\in[0,\alpha]\end{cases}$$
   其中 $\lceil\cdot\rceil$ 和 $\lfloor\cdot\rfloor$ 分别是向上取整和向下取整函数; $Q^{-1}(q_n)=q_n/2^r$
   进一步，实用2-bits 表示符号位 (sign-bits) 。如此，0便有了两种编码方式。

- They present a quantization scheme to encode weight updates as a batch of gradients, which can then be processed by leveraging single instruction multiple data (SIMD) methods. +paillier
- 效果:



Model	Mode	Epochs	Acc./Loss	Time (h)	Traffic (GB)
FMNIST	stock	40	88.62%	122.5	2228.3
	batch	68	88.37%	8.9	58.7
	plain	40	88.62%	3.2	11.17
CIFAR	stock	285	73.97%	9495.6	16422.0
	batch	279	74.04%	131.3	227.8
	plain	285	73.97%	34.2	11.39
LSTM	stock	20	0.0357	8484.4	15347.3
	batch	23	0.0335	105.2	175.9
	plain	20	0.0357	12.3	10.4

- FLASHE: Additively Symmetric Homomorphic Encryption for Cross-Silo Federated Learning

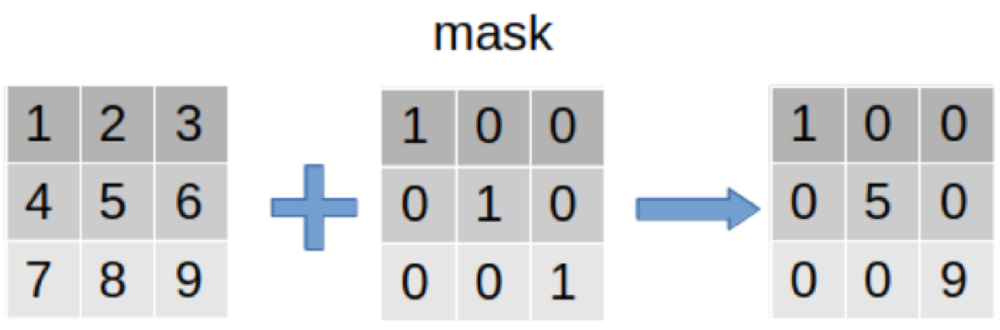
  与上面的paper相比，增加了一些小技巧:

  1. **取消了非对称加密，采用带有随机数的加法同态加密**

  $$\begin{aligned} & E_k(m) = (c, i, \{j\}), \\ s.t.\quad & c_d = (m_d + F_k(i \mid\mid j \mid\mid d) - F_k(i \mid\mid (j+1) \mid\mid d)), \qquad (1) \\ & \text{for } 1 \le d \le D, \end{aligned}$$

  F：随机数发生函数PRF 上面是double-masking版本

  2. **top-*k*稀疏化技术：减少通信成本**

  > Gradient Sparsification：使用梯度大小作为重要程度的简单启发式: **只有梯度大于阈值的才可以发送。** 为了避免信息损失，将剩余的局部梯度累积，直到这些梯度成为一个较大的梯度(达到阈值)再发送。
  >
  > 
  >

（1）当不需要稀疏化时，在采用double masking的情况下进一步优化计算开销。

（2）当使用稀疏化时，自适应地在double masking和single masking之间切换，以获得最佳性能。

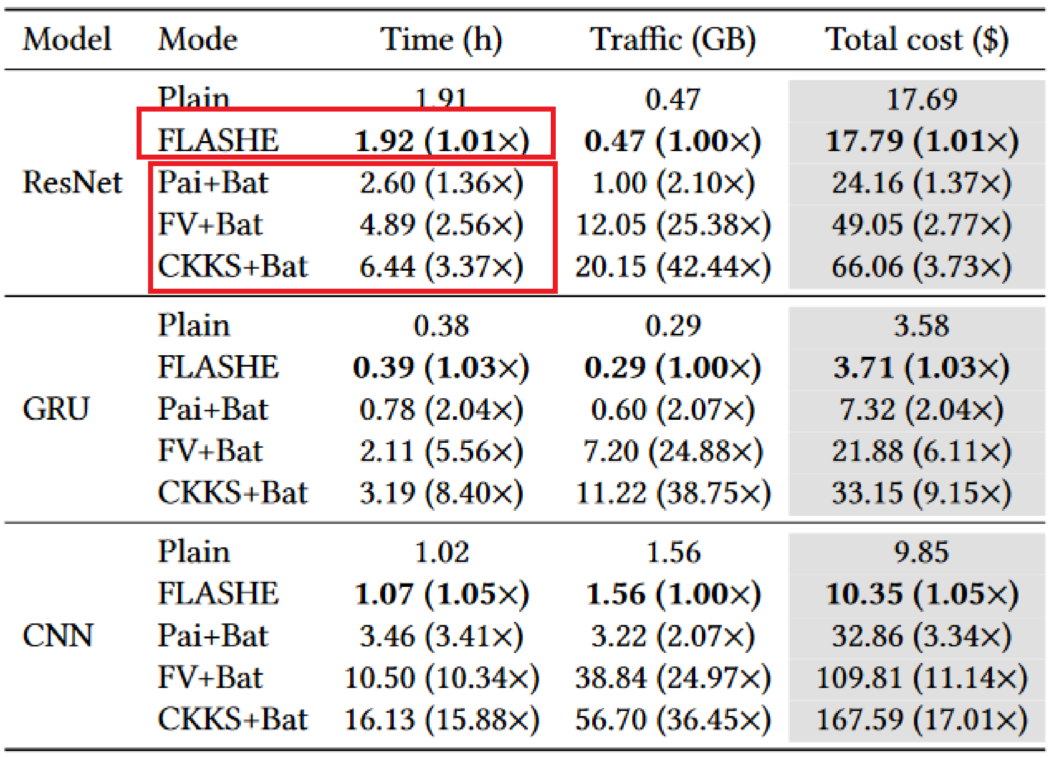
具体地，在每一轮中，在确定了其当前更新的稀疏化掩码sparsification mask之后，每个客户端将该掩码发送到服务器（第21行）。然后，服务器通过首先计算两种方案所需的掩码数，然后选择结果较小的方案，帮助客户端确定是使用double masking 还是single masking（第5行）。该方案不需要任何前提。

> **Theorem 2.** *Given the total number of clients* N*, the number of scalars in a model update* D, and **dropout rate** d *in a federation round, the* **expected total number of masks needed to generate** *in that round for each surviving client under the double masking scheme and single masking scheme are* $2(-Nd^2+(N-1)d+2)D$ and $(-Nd+N+1)D$, respectively.

Alg. 1: Top $s\%$ sparsification with tuned FLASHE.

```
Function ServerCoordinate()
    for $i = 1, 2, \ldots$ do
        Issue ClientTrainIter($i$) at each client $j$
        Collect $M'_{i,j}$ from each client $j$
        Determine the masking scheme and notify clients
        Collect $[[\Delta w'_{i,j}]]$ from each client $j$
        $[[\Delta w'_i]] \leftarrow \sum_j [[\Delta w'_{i,j}]]$, and $M'_i \leftarrow \sum_j M'_{i,j}$
        Dispatch $[[\Delta w'_i]]$ and $[[M'_i]]$ to clients
Function ClientTrainIter($i, j$)
    /* Local training. */
    $w_{i,j} = w_{i-1}$
    for $e = 1, 2, \ldots$ do
        Update $w_{i,j}$ based on local dataset $\chi$
    $\Delta w_{i,j} = w_{i,j} - w_{i-1}$
    /* Per-layer top $s\%$ sparsification with error reserved. */
    $\Delta w_{i,j} = \Delta w_{i,j} + r_{i-1,j}$
    for $l = 1, \ldots, L$ do
        Select threshold: $\theta \leftarrow s\%$ largest in $\Delta w_{i,j}[l]$
        Determine Mask: $M_{i,j}[l] \leftarrow |\Delta w_{i,j}[l]| > \theta$
        $r_{i,j}[l] \leftarrow \Delta w_{i,j}[l] \odot \neg M_{i,j}[l]$
        $\Delta w_{i,j}[l] \leftarrow \Delta w_{i,j}[l] \odot M_{i,j}[l]$
    /* Order permutation. */
    Permutate $\Delta w_{i,j}$ and $M_{i,j}$ consistently, obtain $\Delta w'_{i,j}$ and $M'_{i,j}$
    /* Adaptive masking and synchronization. */
    Send $M'_{i,j}$ to the server, and receive decision from it
    Quantize and encrypt $\Delta w'_{i,j}$ into $[[\Delta w'_{i,j}]]$
    Send $[[\Delta w'_{i,j}]]$ to the server and receive $[[\Delta w'_i]]$ and $M'_i$
    Decrypt and unquantize $[[\Delta w'_i]]$, obtain $\Delta w'_i$
    /* Normalization and order restoration. */
    Normalize $\Delta w'_i$ based on $M'_i$
    Restore $\Delta w_i$ from $\Delta w'_i$
    $w'_i = \Delta w'_i + w'_{i-1}$
```

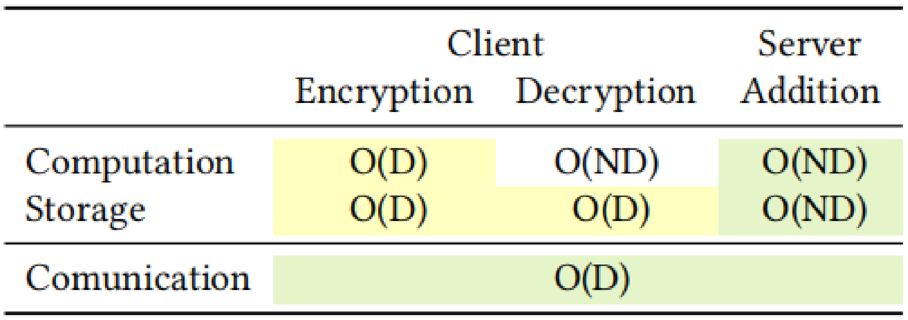
- 效果：与明文训练相比，FLASHE略微增加了训练时间≤6%，且没有引起通信开销



Model	Mode	Time (h)	Traffic (GB)	Total cost ($)
ResNet	Plain	1.91	0.47	17.69
	FLASHE	**1.92 (1.01×)**	**0.47 (1.00×)**	**17.79 (1.01×)**
	Pai+Bat	2.60 (1.36×)	1.00 (2.10×)	24.16 (1.37×)
	FV+Bat	4.89 (2.56×)	12.05 (25.38×)	49.05 (2.77×)
	CKKS+Bat	6.44 (3.37×)	20.15 (42.44×)	66.06 (3.73×)
GRU	Plain	0.38	0.29	3.58
	FLASHE	**0.39 (1.03×)**	**0.29 (1.00×)**	**3.71 (1.03×)**
	Pai+Bat	0.78 (2.04×)	0.60 (2.07×)	7.32 (2.04×)
	FV+Bat	2.11 (5.56×)	7.20 (24.88×)	21.88 (6.11×)
	CKKS+Bat	3.19 (8.40×)	11.22 (38.75×)	33.15 (9.15×)
CNN	Plain	1.02	1.56	9.85
	FLASHE	**1.07 (1.05×)**	**1.56 (1.00×)**	**10.35 (1.05×)**
	Pai+Bat	3.46 (3.41×)	3.22 (2.07×)	32.86 (3.34×)
	FV+Bat	10.50 (10.34×)	38.84 (24.97×)	109.81 (11.14×)
	CKKS+Bat	16.13 (15.88×)	56.70 (36.45×)	167.59 (17.01×)

N the total number of clients　D： thelength of a plaintext vector

Table 3. Cost summary for vanilla FLASHE. Green (yellow) cells indicate optimality in the absolute (asymptotic) sense.



	Client		Server
	Encryption	Decryption	Addition
Computation	O(D)	O(ND)	O(ND)
Storage	O(D)	O(D)	O(ND)
Comunication	O(D)		

- Efficient Batch Homomorphic Encryption for **Vertically Federated XGBoost** 2021

  受BatchCrypt启发，设计了一种新颖的批处理方法来解决 允许相当少的负数数量的限制。将加密相关的计算和传输成本减少了近一半；编码过程包括四个步骤，包括移位、截断、量化和批处理

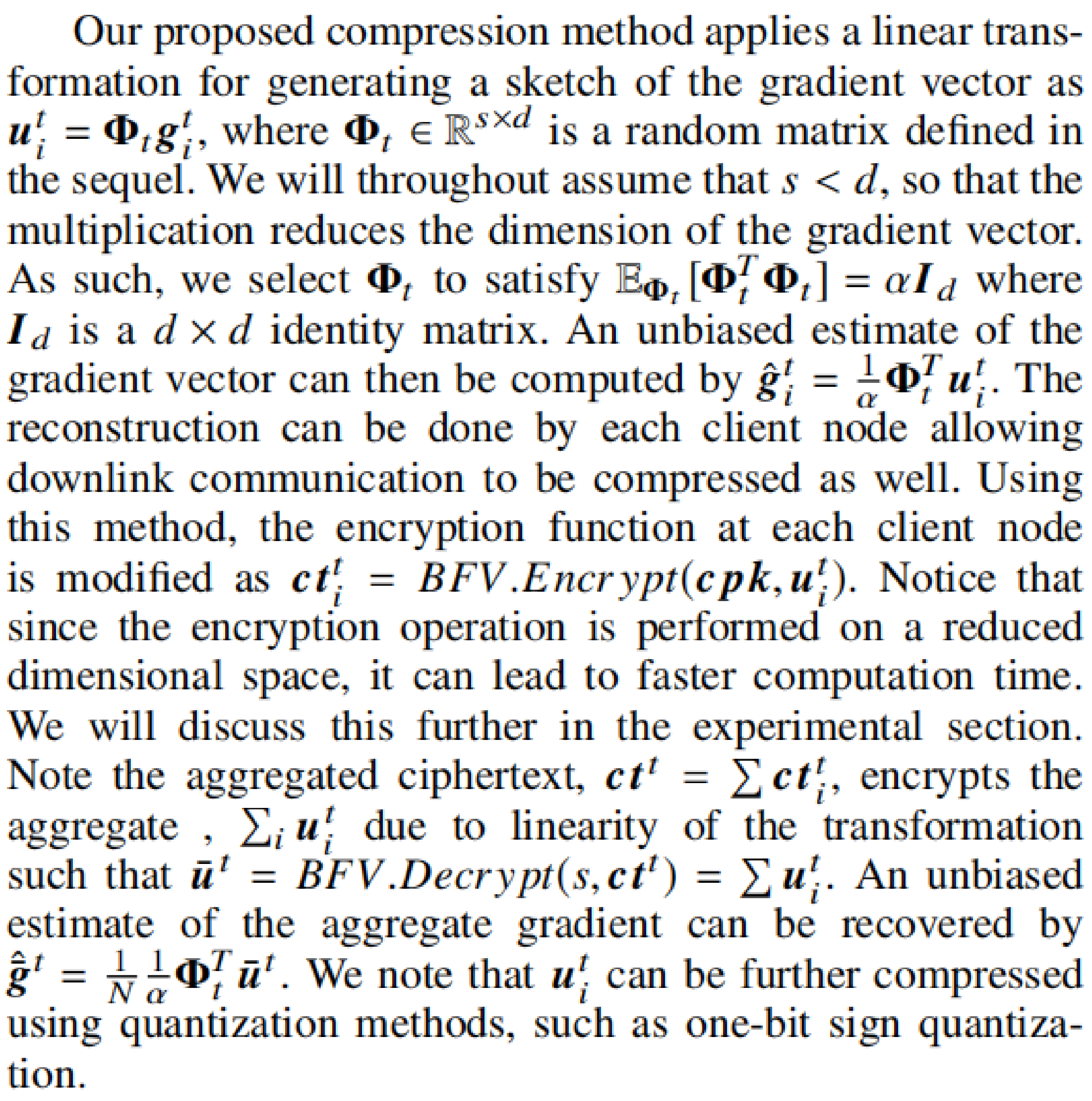
- VerifyNet: Secure and Verifiable Federated Learning **CCFA 2019**

  first propose a **double-masking protocol** to guarantee the confidentiality of users' local gradients during the federated learning. 有安全性证明

- Secure Aggregation in Federated Learning via Multiparty Homomorphic Encryption ccfc [2021 IEEE Globecom Workshops]

  BFV Encryption同态支持+，*

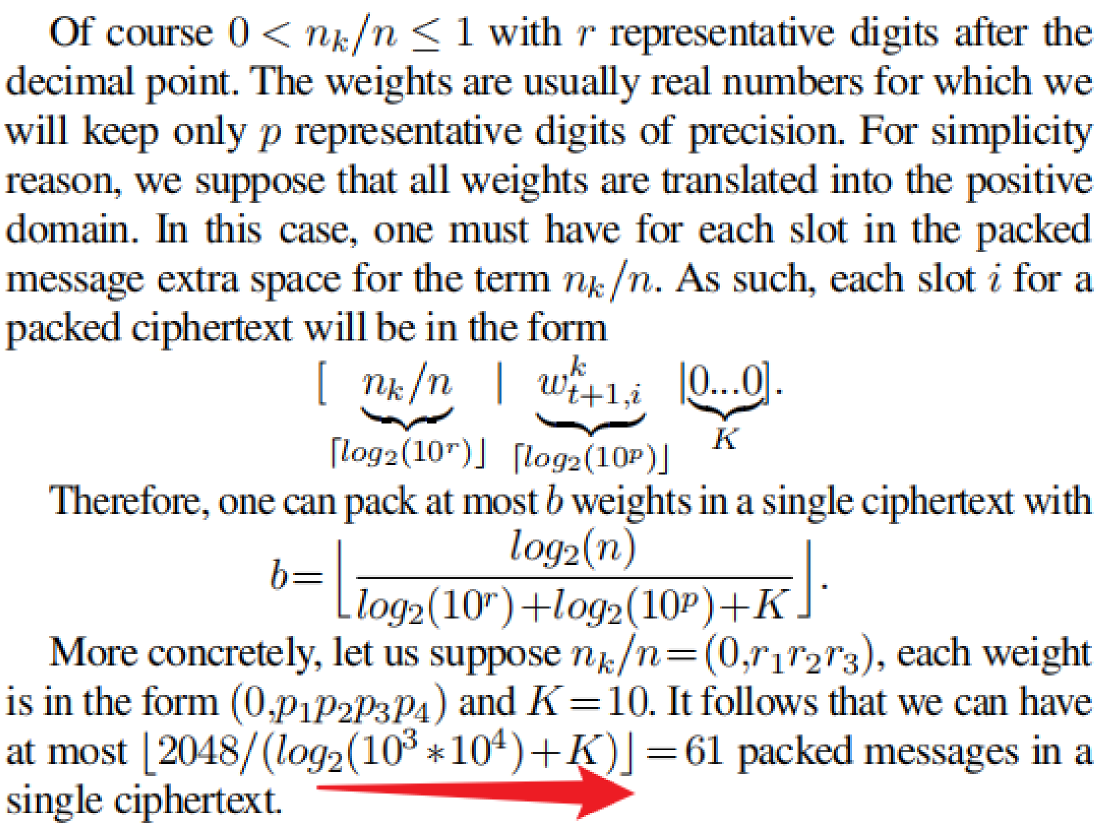
  使用了一个不同的**Gradient Compression**技术（popular: top-*k*, rand-*k* and sketching based methods ） 加速：



Our proposed compression method applies a linear transformation for generating a sketch of the gradient vector as $\boldsymbol{u}_i^t = \boldsymbol{\Phi}_t \boldsymbol{g}_i^t$, where $\boldsymbol{\Phi}_t \in \mathbb{R}^{s \times d}$ is a random matrix defined in the sequel. We will throughout assume that $s < d$, so that the multiplication reduces the dimension of the gradient vector. As such, we select $\boldsymbol{\Phi}_t$ to satisfy $\mathbb{E}_{\boldsymbol{\Phi}_t}[\boldsymbol{\Phi}_t^T \boldsymbol{\Phi}_t] = \alpha \boldsymbol{I}_d$ where $\boldsymbol{I}_d$ is a $d \times d$ identity matrix. An unbiased estimate of the gradient vector can then be computed by $\hat{\boldsymbol{g}}_i^t = \frac{1}{\alpha} \boldsymbol{\Phi}_t^T \boldsymbol{u}_i^t$. The reconstruction can be done by each client node allowing downlink communication to be compressed as well. Using this method, the encryption function at each client node is modified as $\boldsymbol{ct}_i^t = BFV.Encrypt(\boldsymbol{cpk}, \boldsymbol{u}_i^t)$. Notice that since the encryption operation is performed on a reduced dimensional space, it can lead to faster computation time. We will discuss this further in the experimental section. Note the aggregated ciphertext, $\boldsymbol{ct}^t = \sum \boldsymbol{ct}_i^t$, encrypts the aggregate , $\sum_i \boldsymbol{u}_i^t$ due to linearity of the transformation such that $\bar{\boldsymbol{u}}^t = BFV.Decrypt(s, \boldsymbol{ct}^t) = \sum \boldsymbol{u}_i^t$. An unbiased estimate of the aggregate gradient can be recovered by $\hat{\bar{\boldsymbol{g}}}^t = \frac{1}{N} \frac{1}{\alpha} \boldsymbol{\Phi}_t^T \bar{\boldsymbol{u}}^t$. We note that $\boldsymbol{u}_i^t$ can be further compressed using quantization methods, such as one-bit sign quantization.

- A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing[2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)]

  Batching for Paillier + Quantization



Of course $0 < n_k/n \leq 1$ with r representative digits after the decimal point. The weights are usually real numbers for which we will keep only p representative digits of precision. For simplicity reason, we suppose that all weights are translated into the positive domain. In this case, one must have for each slot in the packed message extra space for the term n_k/n. As such, each slot i for a packed ciphertext will be in the form

$$[\ \underbrace{n_k/n}_{\lceil log_2(10^r)\rceil}\ \mid\ \underbrace{w^k_{t+1,i}}_{\lceil log_2(10^p)\rceil}\ |\underbrace{0...0}_{K}].$$

Therefore, one can pack at most b weights in a single ciphertext with

$$b = \left\lfloor \frac{log_2(n)}{log_2(10^r)+log_2(10^p)+K} \right\rfloor.$$

More concretely, let us suppose $n_k/n = (0,r_1r_2r_3)$, each weight is in the form $(0,p_1p_2p_3p_4)$ and $K = 10$. It follows that we can have at most $\lfloor 2048/(log_2(10^3 * 10^4)+K)\rfloor = 61$ packed messages in a single ciphertext.

- POSEIDON: Privacy-Preserving Federated Neural Network Learning **NDSS 2021** 6C-1 没有引用BatchCrypt&FLASHE

  这篇文章主要是扩展了SPINDLE方案，将线性模型扩展到神经网络模型，提出了一个新的系统POSEIDON。具体地，文章首先基于2017年提出的CKKS方案设计了多方同态加密，虽然并未给出具体的基于CKKS的多方同加密方案内容，但这很容易从2020年基于BFV的MHE(多方同态加密)进行推导。此外，文章的贡献还包括1）设计了一种交替打包方式，可以有效地对加密数据使用单指令多数据 (SIMD) 操作；减少了矩阵向量乘法中的转置和旋转操作2）给出了神经网络模型中密码参数和机器学习参数的联系，构建了一个优化模型用来减少密码操作。

  达到的效果： POSEIDON能够达到与集中式训练或不保障隐私的分布式训练方法类似的精度，并且**其计算和通信开销与参与方数目呈线性扩展。** POSEIDON在MNIST数据集上训练了一个3层神经网络，有784个特征和60K样本，分布在10个参与方中，训练时间不到2小时。

  > The execution times produced by Nandakumar et al. [85] for processing one batch of 60 samples in a single thread and 30 threads for a NN structure with *d*=64, *h*1=32, *h*2=16, *h*3=2, are respectively **33,840s and 2,400s**. When weevaluate the same setting, but with *N* =10 parties, we observe that POSEIDON processes the same batch in **6.3s and 1s,** respectively. We also achieve tronger security guarantees (128 bits) than [85] (80 bits). Finally, for a NN structure with 2-hidden layers of 128 neurons each, and the MNISTdataset, CryptoDL [51] processes a batch with *B*=192 in **10,476.3s**, whereas our system in the distributed setting processes the same batch in **34.7s**. Therefore, POSEIDON is the only solution that performs both training and inference of NNs in an *N*-party setting, yet protects data and model confidentiality withstanding collusions up to *N* −1 parties.

  - tree型拓扑（通信开销小） centralized federated learning topology where all learners interact directly with a central controller.。文中采用MapReduce架构来描述协作训练，根节点P1相当于聚合器。

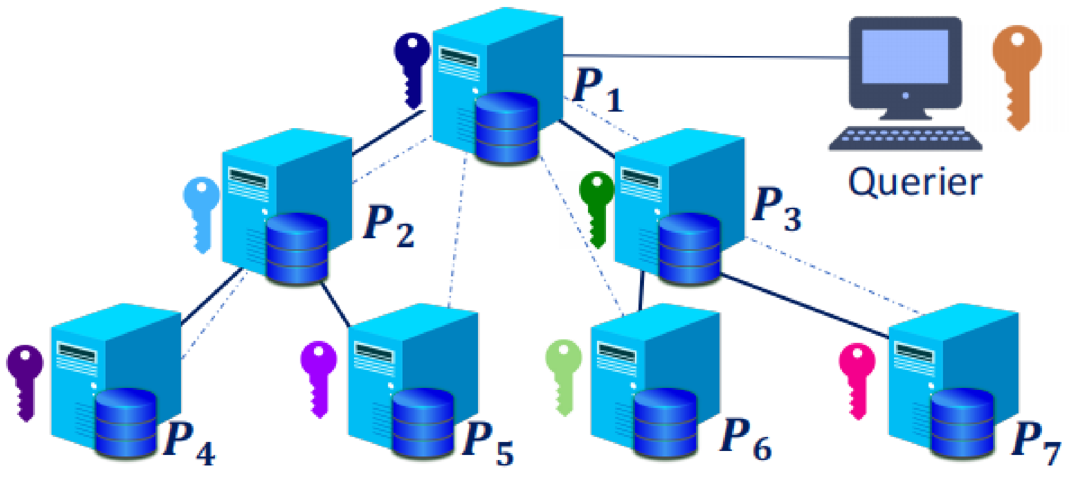


Figure 1: POSEIDON's *System Model.*

梯度下降挑战：
1.非多项式激活函数，如Sigmoid, Softmax
2.昂贵的同态运算，即旋转，自举
3.特定于模型的函数，即卷积神经网络(CNNs)中的池化

对应措施：
1.激活函数的最小二乘近似
2.启用单指令多数据(SIMD)的特定问题打包方案
3.引入多个函数到分布式bootstrapping中

Bootstrapping挑战：该模型在多次迭代过程中是持久的→大的乘法深度→密文需要bootstrapping

对应措施：1.高效和写作的bootstrapping 2.通过参数最小化bootstrapping数量

参数化挑战：学习参数和密码参数之间的强相关性

对应措施：选择密码参数的约束优化问题

(334条消息) 论文学习笔记 POSEIDON: Privacy-Preserving Federated Neural Network Learning_JiangChSo的博客-CSDN博客

  - 缺点：the local training done by individual learners is encrypted增加了额外开销
- EaSTFLy: Efficient and secure ternary federated learning Computers & Security2020 CCFB

  secret sharing+SIMD

```
Algorithm 2: Batching Encode.
    Input: Vector of ternary gradients $\tilde{\mathbf{g}}$, $log_2(2N+1)$-bit prime $P$
    Output: Batching encoded gradients $\bar{\mathbf{g}}$
  1 foreach $g$ in $\tilde{\mathbf{g}}$ do
  2     Encode $g$ as
  3     $g = g \mod P$;
  4 for element in $\tilde{\mathbf{g}} \mod P$ do
  5     Encode $\Delta$ elements into one plaintext $\mathcal{M}$,
  6     where $\Delta = \lfloor \frac{\lfloor log_2 N \rfloor}{prec+pad} \rfloor$;
  7 Denote all $\mathcal{M}$s as vector $\bar{\mathbf{g}}$.
  8 return $\bar{\mathbf{g}}$.
```

```
Algorithm 3: Batching Decode.
    Input: Batching encoded gradients $\bar{\mathbf{g}}$, $log_2(2N+1)$-bit prime
           $P$
    Output: Vector of $log_2(2N+1)$-bit gradients $\tilde{\mathbf{g}}$
  1 foreach $\mathcal{M}$ in $\bar{\mathbf{g}}$ do
  2     for element in $\mathcal{M}$ do
  3         if $0 \le element \le \lfloor P/2 \rfloor$ then
  4             $element = element$
  5         else if $\lfloor P/2 \rfloor + 1 \le element < P$ then
  6             $element = element - P$
  7         else
  8             Error
  9 Denote all decode elements as vector $\tilde{\mathbf{g}}$
 10 return $\tilde{\mathbf{g}}$
```

- Homomorphic Encryption-based Privacy-preserving Federated Learning in IoT-enabled Healthcare System ([IEEE Transactions on Network Science and Engineering]() 2022CCFnone)

  用基于数据质量的加权平均算法来代替传统的基于数据量的权重计算方法。

  data quality定义

$$Q_i^t = \frac{\delta}{\left\|\mathbf{g}_i^t - \mathbf{g}_{global}^{t-1}\right\|_2} = \frac{\delta}{\sum_{d=1}^{|\mathbf{g}|}\left(g_{i-d}^t - g_{global-d}^{t-1}\right)^2}$$

  结合了MPC：Diffifie-Hellman key exchange and Shamir secret sharing algorithm；Elgamal加法同态

  Computation cost of client: $O(ND)$

  Computation cost of server: $O(NDm)$　m：the number of clients dropping out 若没有客户端退出，为$O(N)$

- Privacy-Preserving Federated Learning Using Homomorphic Encryption（Applied Sciences, 2022 - mdpi ccfnone）

所有的client公用一对密钥的泄露风险比较大，这篇使用了Distributed homomorphic cryptosystem (DHC)，split 私钥避免了这一问题。

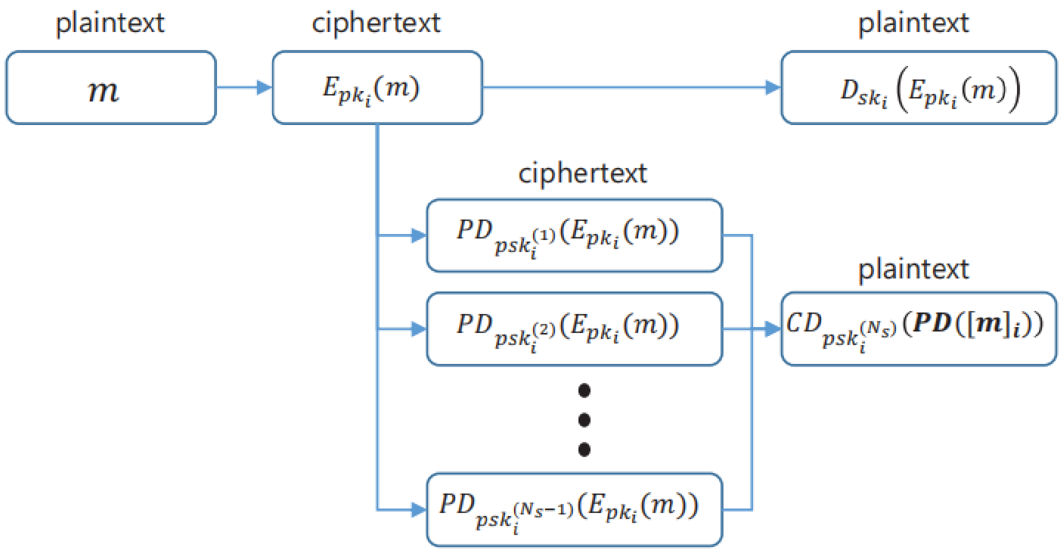

plaintext
m
ciphertext
$E_{pk_i}(m)$
plaintext
$D_{sk_i}\left(E_{pk_i}(m)\right)$
ciphertext
$PD_{psk_i^{(1)}}(E_{pk_i}(m))$
$PD_{psk_i^{(2)}}(E_{pk_i}(m))$
$PD_{psk_i^{(N_S-1)}}(E_{pk_i}(m))$
plaintext
$CD_{psk_i^{(N_S)}}(\boldsymbol{PD}([\mathbf{m}]_i))$

Figure 1. Diagram for encryption, decryption, and partial decryption.

> **阈值同态加密算法**中存在多个私钥、一个（或多个）公钥，使用该公钥系统加密的密文之间可以相互计算，并且只有当参与解密的私钥数量达到一定阈值时，才能成功解密密文，所以这种多密钥同态加密算法又被称为阈值同态加密。
>
> Multiparty Homomorphic Encryption (MHE)* C. Mouchet, J. R. Troncoso-pastoriza, J.-P. Bossuat, and J. P. Hubaux. Multiparty homomorphic encryption: From theory to practice. In *Technical Report* [*https://eprint.iacr.org/2020/304*](https://eprint.iacr.org/2020/304), 2019.
>
> *IBMHOM* is based on the *t*-threshold Paillier cryptosys-tem, that is, the ciphertext must be decrypted with more than *t* secret keys.

HE+DNN

2022-Sphinx: Enabling Privacy-Preserving Online Learning over the Cloud(S&P)

Sphinx 的核心是将深度神经网络中的每个线性层分为两部分：线性分量 W 和偏置分量 b。 *Sphinx 使用同态加密 (HE) 加密所有偏置分量，并使用差分隐私 (DP) 扰乱线性分量。*我们表明，这种设计能够实现我们的高吞吐量训练和低延迟推理协议（第 V 节），并从理论和实验的角度建立 DP 和 HE 技术之间的互惠关系。为了加速 HE 方案下的训练过程，Sphinx 进行了多项系统优化（第 VII 节）。首先，*通过刻意设计特征、梯度和模型参数之间的同态算术运算行为，它避免了大多数用于密文乘法的昂贵的重新缩放和重新线性化操作*。其次，Sphinx加速了加密操作，减小了密文大小，进一步降低了客户端和服务器之间的通信时间.

其他HE+ML相关

2022-SIMC: ML Inference Secure Against Malicious Clients at Semi-Honest Cost(USENIX)

神经网络由两种类型的层或函数组成：线性层（包括矩阵乘法、卷积等函数）和非线性层（包括 ReLU、ReLU6、Maxpool 等函数）。在 MUSE 考虑的基准测试中，MUSE 近 99% 的通信开销（以及大约 80% 的整体性能开销）是由于非线性层的安全计算协议造成的。 SIMC 的核心是一种用于安全计算非线性层的全新协议，与 MUSE 在计算和通信方面相比，它的分析重量甚至更轻。MUSE 使用计算量大的同态加密 以及通信量大的经过身份验证的 Beaver 三元组 来实现它们的非线性层。相比之下，SIMC 使用廉价的不经意传输和一次性加密来完成相同的任务。

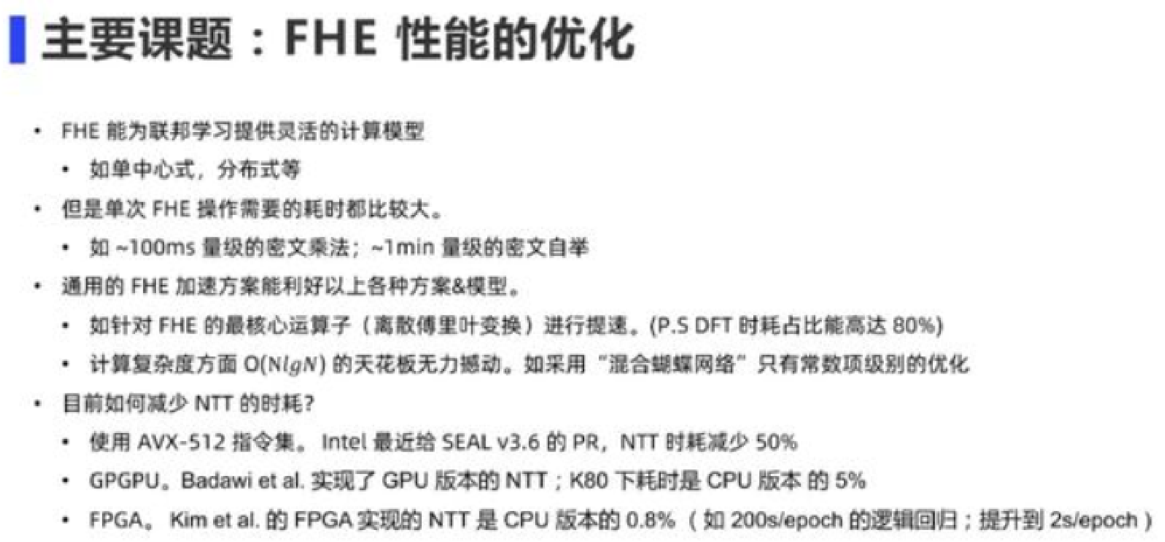
2022-Privacy-Preserving and Verifiable Convolutional Neural Network Testing

等[隐私保护-文献整理 - 知乎 (zhihu.com)]

FHE性能提升

- PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption S&P 2021

  提出PEGASUS 框架新方案， 能够在 FHE 密文上直接计算非线性函数。如FHE.Enc(x) -> FHE.Enc(σ(x))。首先对输入空间离散化，然后进行计算。但是由于FHE 自身存在噪音，函数越光滑，误差就越小。它的优点是灵活性好，无须额外的通信，缺点是需要的计算量较大（0.5s -- 1s 一次)。



主要课题：FHE 性能的优化

- FHE 能为联邦学习提供灵活的计算模型
  - 如单中心式，分布式等
- 但是单次 FHE 操作需要的耗时都比较大。
  - 如 ~100ms 量级的密文乘法；~1min 量级的密文自举
- 通用的 FHE 加速方案能利好以上各种方案&模型。
  - 如针对 FHE 的最核心运算子（离散傅里叶变换）进行提速。(P.S DFT 时耗占比能高达 80%)
  - 计算复杂度方面 $O(N lg N)$ 的天花板无力撼动。如采用 “混合蝴蝶网络” 只有常数项级别的优化
- 目前如何减少 NTT 的时耗?
  - 使用 AVX-512 指令集。 Intel 最近给 SEAL v3.6 的 PR，NTT 时耗减少 50%
  - GPGPU。Badawi et al. 实现了 GPU 版本的 NTT；K80 下耗时是 CPU 版本 的 5%
  - FPGA。 Kim et al. 的 FPGA 实现的 NTT 是 CPU 版本的 0.8%（如 200s/epoch 的逻辑回归；提升到 2s/epoch）

Q：阿里的全同态是基于微软的seal还是自己实现的?

A：基于seal，算法基础逻辑不做改变，只是对算法的性能进行了一些优化。

Q：全同态和fate的半同态效果怎么样?

A：**如果只是算两个密文加法fate半同态更快。如果是更大矩阵向量乘法，有办法做到比半同态快。**

Q：加密数据上传，网络消耗是否很大?

A：网络消耗很大，耗时取决数据和计算模型。线性模型比多轮通信会少一点。

Q：半同态不能支持sigmoid，全同态不支持sigmoid，遇到逆运算、对数运算、除法运算怎么办?

A：参考我们发的论文，提供不需要解密直接在全同态密文上计算非线性函数。有办法解决只是性能比较慢。

Q：请问全同态联邦学习可以支撑多大数据联邦建模，有实际应用吗?

A：取决于单点多强大。我们用的是一台，实际上也可以用多台机器组成。多个同态密文，可以使用MapReduce，调度在工程上可以有优化。

[阿里联邦学习新方式：全同态密码 - 知乎 (zhihu.com)]

其他

其他HE：

FV： 支持+ *

CKKS：支持+ *；可以加密实数或复数，但只能得到近似的结果。

Scheme	Supported Function	Plaintext	Semantic Security	Supporting Library
Paillier [52]	$+$	integer	✓	python-paillier [1]
FV [33]	$+, \times$	integer	✓	SEAL [12], PALISADE [9]
CKKS [27]	$+, \times$	real/complex numbers	✓	SEAL, PALISADE, HElib [7] , HEAAN [6]