SecretFlow-SPU: A Performant and User-Friendly Framework for Privacy-Preserving Machine Learning

**

此次介绍是蚂蚁隐语团队Junming Ma和Yancheng Zheng等人发表在USENIX ATC'2023的SecretFlow-SPU，论文链接如下：
https://www.usenix.org/conference/atc23/presentation/ma
**

1. 背景知识

隐私保护机器学习近年来获得了学术界和工业界的广泛关注，但是设计一个高效的方案是一件十分困难且繁琐的事情。研究者既需要深入了解底层安全多方计算协议算法，也需要熟练掌握上层机器学习应用技术。虽然已有的MP-SPDZ、EzPC、Crypten和TF-Encrypted从不同的角度推进该领域的发展，但是目前各个方案均有自己的短板。SecretFlow-SPU（下文简称SPU）提出了PPHLO（Privacy-Preserving High-Level Operations）中间层表示将机器学习和安全多方计算链接起来，提供了高效、友好的开发框架。具体来说，SPU的主要贡献如下：

- SPU支持多种主流机器学习框架，可以大大加速开发、调试和部署隐私保护机器学习的时间周期。
- 本文提出了面向MPC的中间表示PPHLO链接机器学习和安全多方计算协议设计，同时本文提出了多项编译层面的优化技术加速PPHLO在执行时的效率。
- 本文在多种实验设置下评估了性能，实际效率大大优于已有类似方案，且SPU更加用户友好。

2. 系统架构

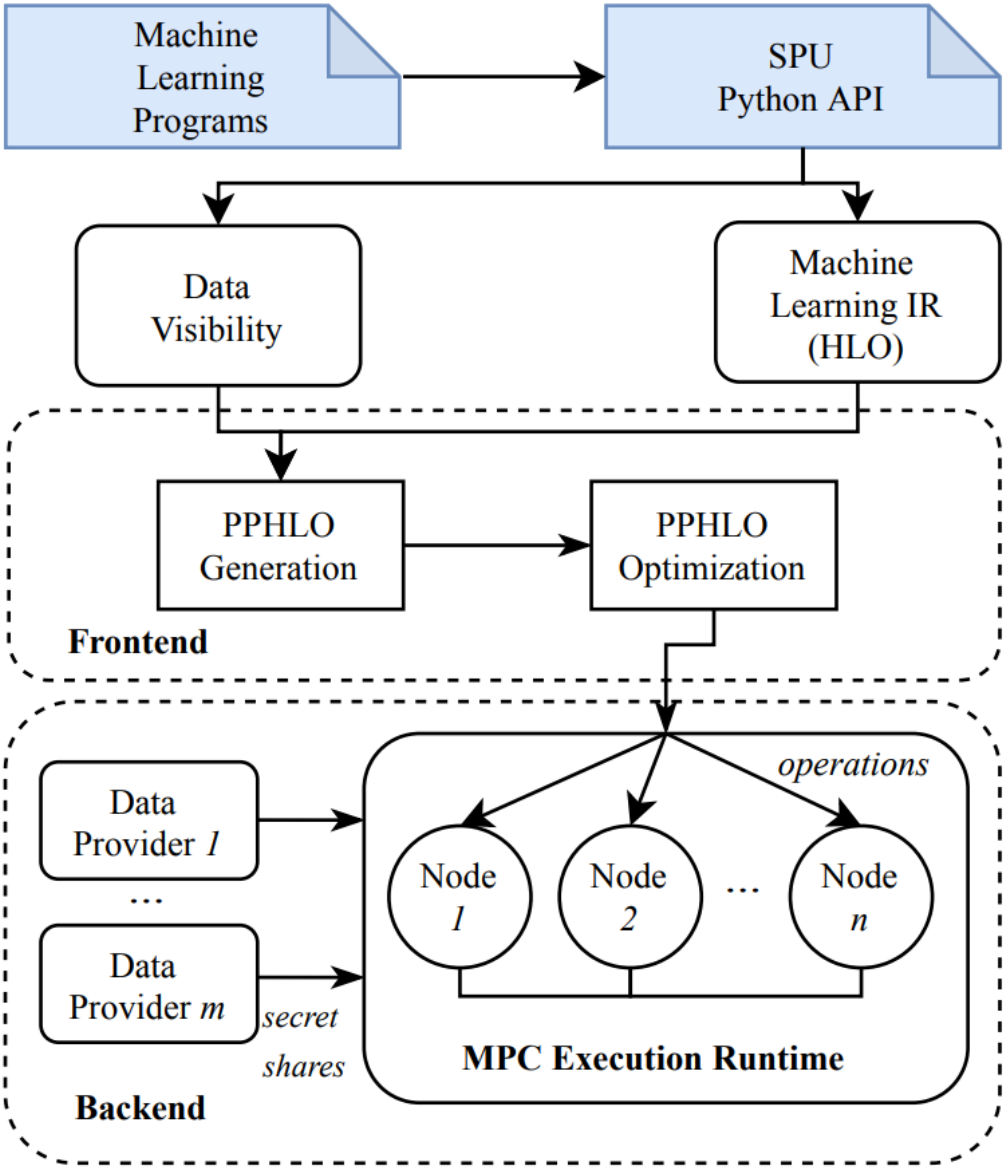

Figure 1: SPU architecture.

如上图所示，SPU接收明文模型的中间表示（HLO）和数据可见性标识，在Frontend层首先生成PPHLO，进一步兑PPHLO进行针对MPC协议特点的编译层优化；完成之后，Backend层接收优化后的PPHLO，并将其分发到MPC协议的执行节点进行协议执行。同时，数据提供方提供各自的秘密数据。

为了表示不同的参数方的数据可见，SPU使用修饰器'@ppd.device()'来指定可见方。如下图中alice和bob分别对应P1和P2。同时'@ppd.device("SPU")'指定需要调用MPC协议计算的函数。

```python
import jax.numpy as jnp
import numpy as np
import spu.binding.util.distributed as ppd

# init SPU backend nodes
with open("/path/to/config", 'r') as file:
    conf = json.load(file)
ppd.init(conf["nodes"], conf["devices"])

# specify data visibility
@ppd.device("P1")
def data_from_alice():
    return np.random.randint(100, size=(4,))

# specify data visibility
@ppd.device("P2")
def data_from_bob():
    return np.random.randint(100, size=(4,))

# specify a private function
@ppd.device("SPU")
def compare(x, y):
    return jnp.maximum(x, y)

# x & y will be automatically
# fetched by SPU (as secret shares)
x = data_from_alice()
y = data_from_bob()

# compare will be evaluated privately by SPU
z = compare(x, y)

# reveal the real value of z
print(f"z = {ppd.get(z)}")
```

而在PPHLO中的张量包含三个属性：<Shape, Data Type, Visibility>，其中Shape是表示张量维度，是公开的；Data Type是数据类型，分为boolean，integer和fixed-point；而Visibility则是PPHLO中独有的属性，分为secret和public两种情况。在PPHLO的静态表示中，各个属性的表示情况如下图所示：

```
func.func @main(%arg0: tensor<4x!pphlo.sec<i32>>, %arg1: tensor<4x!pphlo.sec<i32>>)
    -> tensor<4x!pphlo.sec<i32>> {
  %0 = "pphlo.greater"(%arg0, %arg1) : (tensor<4x!pphlo.sec<i32>>, tensor<4x!pphlo.sec<i32>>)
    -> tensor<4x!pphlo.sec<i1>>
  %1 = "pphlo.select"(%0, %arg0, %arg1) : (tensor<4x!pphlo.sec<i1>>, tensor<4x!pphlo.sec<i32>>,
    tensor<4x!pphlo.sec<i32>>) -> tensor<4x!pphlo.sec<i32>>
  return %1 : tensor<4x!pphlo.sec<i32>>
}
```

例如，%arg0: tensor<4x!pphlo.sec>表示参数0是一个4维秘密张量，数据类型是32比特整数。而协议内各个参数、算子协议之间的关系也可以很容易从上图中分析出来，大大方便了调试。

3. Frontend设计

给定机器学习程序的HLO和初始数据可见性，Frontend首先生成PPHLO，进一步调用多项优化来提升MPC协议执行时的效率。具体来说，SPU目前支持的优化如下：

- Mixed-data-type multiplication fusion：乘法门的输入是一个整数和一个定点数时，可以节约一次截断；
- Mixed-visibility multiplication operands reorder：对于secret×pub×pub的连续乘法，可以更改顺序为pub×pub×secret，这样两个公开数的结果可以执行本地截断，从而节省一次安全截断操作；
- Inverse square root transformation：对于操作$y/(\sqrt{(}x)+u)$，鉴于$\mathrm{rsqrt}()$比$\sqrt{()}$和求倒数更加高效，可以将其替换为$y\cdot\mathrm{rsqrt}(x+eps())$。
- Select predicate reuse：选择门$\mathrm{Select}(a,b,pred)=b+pred\cdot(a-b)$，当$pred=1$时返回$a$；返回0。其中，$pred$通常是布尔运算结果（布尔分享），因此需要将$pred$转化为算术分享支持上述操作。在面临针对同一个$pred$的多个选择门的时候，可以将$pred$首先转化为算术分享，然后在多个选择门中复用。
- Max-pooling transformation：明文下的最大池化操作在前向计算中计算最大值；反向传播时将最大值所在置为梯度，其余位置设为0。可见，反向传播中也包含求最大值的操作。为了节省开销，在前向计算中不仅求出最大值，同时记录其索引，以便在反向传播中使用。

4. Backend设计

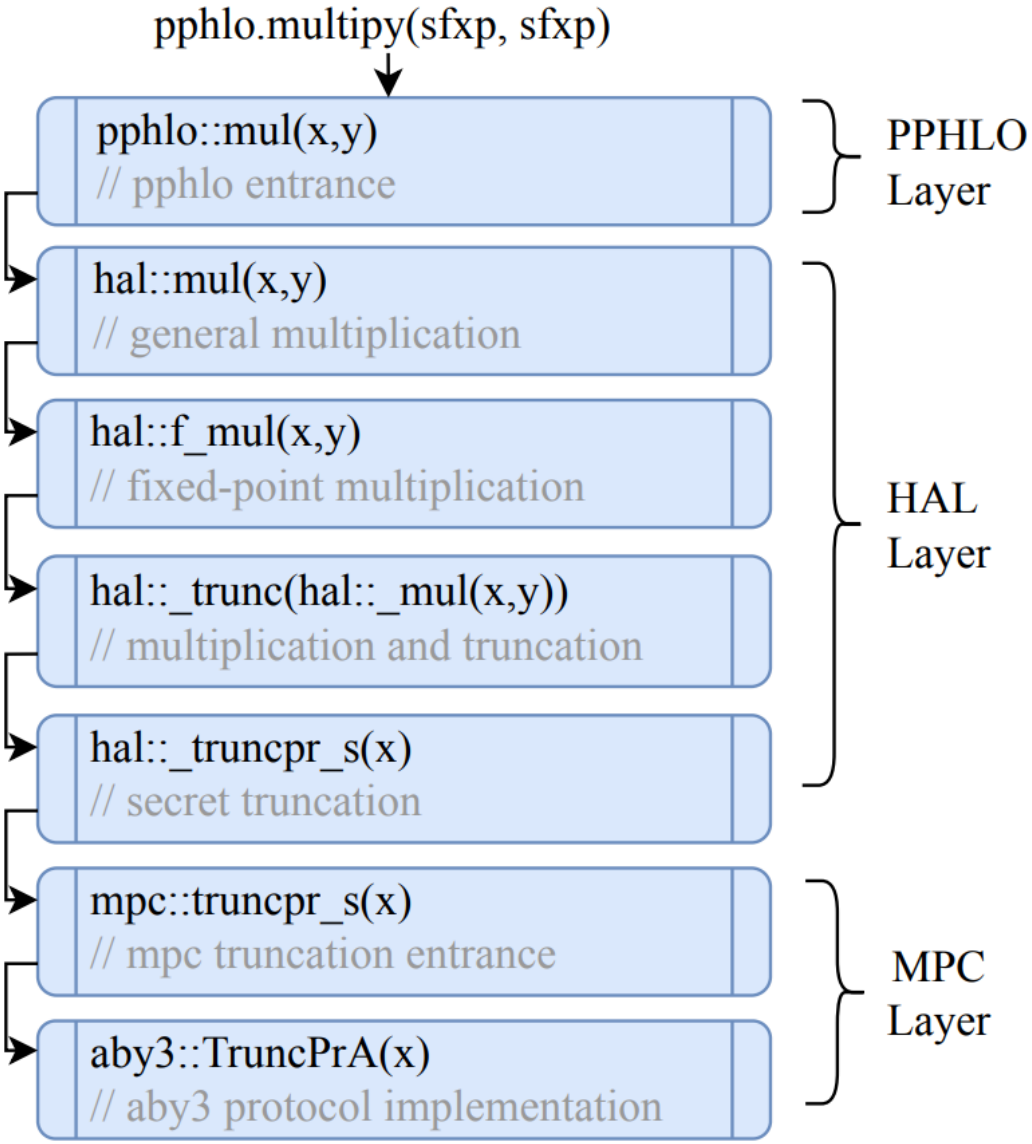

Figure 9: The dispatching path from a PPHLO operation to an MPC protocol in SPU. Different protocols can reuse the same PPHLO/HAL layer code and diverge at the final MPC layer.

如上图所示，SPU在接收PPHLO层的表示之后，将进一步调用底层应用，通过HAL最终到达底层的MPC层实现。在新加入一个协议的时候，安全协议研究者只需要关注在MPC层对应协议的接口实现即可。

另外，SPU还加入了向量化、流式和算子内-算子间同步并行，以尽可能加速协议执行。

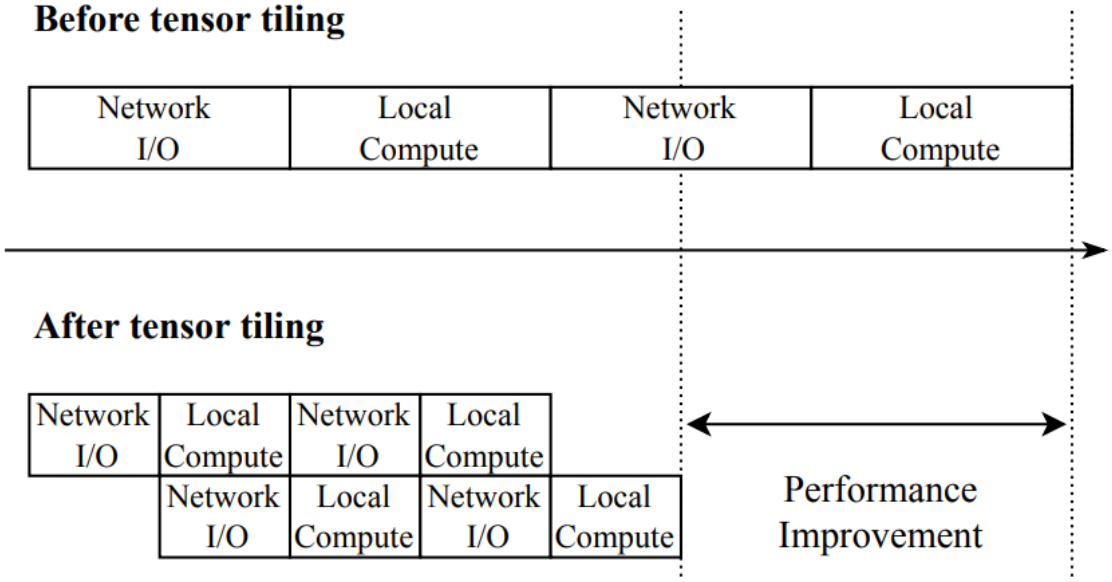

Figure 10: Streaming for MPC operations.

5. 实现测评

本文针对SPU做了多项实验，并和MP-SPDZ、Crypten和TF-Encrypt进行了对比。其中在3方计算下的准确率和时间对比开销如下：

Table 1: The accuracy and seconds per batch of training four neural network models on the MNIST dataset with SGD/Adam/AMSGrad optimizer in four MPC-enabled PPML frameworks. M, T, C, and S are abbreviations of MP-SPDZ [29], TF Encrypted [14], CrypTen [33], and our SPU, respectively. CrypTen does not support Adam and AMSGrad as of the time we write this paper.

Network	Accuracy				Seconds per Batch (LAN)				Seconds per Batch (WAN)			
	M	T	C	S	M	T	C	S	M	T	C	S
A (SGD)	96.8%	96.4%	92.7%	**96.9%**	0.16	0.19	1.43	**0.12**	8.94	**4.60**	58.68	**4.60**
A (Adam)	**97.5%**	97.2%	N/A	97.4%	0.42	0.56	N/A	**0.39**	17.72	12.60	N/A	**7.67**
A (AMSGrad)	**97.6%**	97.4%	N/A	97.5%	0.42	0.71	N/A	**0.41**	18.28	13.26	N/A	**7.68**
B (SGD)	98.1%	98.3%	96.5%	**98.4%**	**1.00**	4.82	25.62	1.04	34.70	15.66	230.15	**9.87**
B (Adam)	97.9%	**98.7%**	N/A	**98.7%**	1.13	4.90	N/A	**1.12**	44.92	18.18	N/A	**11.15**
B (AMSGrad)	98.7%	**98.8%**	N/A	98.6%	1.13	4.78	N/A	**1.12**	45.73	18.08	N/A	**11.23**
C (SGD)	98.5%	**98.9%**	97.3%	98.8%	2.10	7.23	34.06	**1.81**	50.05	22.41	272.11	**12.98**
C (Adam)	98.8%	**99.0%**	N/A	98.9%	2.92	8.33	N/A	**2.37**	67.03	49.51	N/A	**22.87**
C (AMSGrad)	**99.2%**	98.9%	N/A	99.1%	2.94	8.93	N/A	**2.37**	67.49	51.06	N/A	**22.53**
D (SGD)	97.0%	**97.6%**	95.7%	97.2%	0.23	0.39	1.77	**0.22**	11.20	5.35	59.44	**4.89**
D (Adam)	97.8%	**98.0%**	N/A	97.7%	0.45	0.69	N/A	**0.43**	19.87	12.12	N/A	**7.66**
D (AMSGrad)	**98.3%**	97.5%	N/A	97.9%	0.45	0.81	N/A	**0.43**	20.42	12.76	N/A	**7.66**

更多的实验和分析请参考原文。