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

Junming Ma; Yancheng Zheng; Jun Feng; Derun Zhao; Haoqi Wu; Wenjing Fang; Jin Tan; Chaofan Yu; Benyu Zhang; and Lei Wang

2023 USENIX Annual Technical Conference

Security and Privacy
https://www.usenix.org/conference/atc23/presentation/ma
https://github.com/secretflow/spu

Abstract

With the increasing public attention to data security and privacy protection, privacy-preserving machine learning (PPML) has become a research hotspot in recent years. Secure multi-party computation (MPC) that allows multiple parties to jointly compute a function without leaking sensitive data provides a feasible solution to PPML. However, developing efficient PPML programs with MPC techniques is a great challenge for users without cryptography backgrounds.
Existing solutions require users to make efforts to port machine learning (ML) programs by mechanically replacing APIs with PPML versions or rewriting the entire program. Different from the existing works, we propose SecretFlow-SPU, a performant and user-friendly PPML framework compatible with existing ML programs. SecretFlow-SPU consists of a frontend compiler and a backend runtime. The frontend compiler accepts an ML program as input and converts it into an MPC-specific intermediate representation. After a series of delicate code optimizations, programs will be executed by a performant backend runtime as MPC protocols. Based on SecretFlow-SPU, we can run ML programs of different frameworks with minor modifications in a privacy-preserving manner.
We evaluate SecretFlow-SPU with state-of-the-art MPC-enabled PPML frameworks on a series of ML training tasks. SecretFlow-SPU outperforms these works for almost all experimental settings (23 out of 24). Especially under the wide area network, SecretFlow-SPU is up to 4.1× faster than MP-SPDZ and up to 2.3× faster than TF Encrypted.

Problem Statement and Research Objectives

- Privacy-preserving machine learning (PPML) has been gaining popularity due to the pervasive usage of machine learning (ML) and attendant privacy problems. Secure multi-party computation (MPC), a cryptographic technique that enables multiple parties to jointly compute a function without leaking each party's private inputs, brings a provable and practical solution to ML users with strong privacy concerns.
- However, incorporating MPC techniques in ML applications introduces great challenges due to the natural differences between these two fields.
  - MPC experts mainly focus on designing performant cryptographic protocols for low-level computation primitives.
  - In contrast, ML practitioners are more accustomed to constructing high-level ML models using user-friendly frameworks that encapsulates commonly-used ML building blocks.

Proposed Method

- *Can we efficiently run ML programs of mainstream frameworks in a privacy-preserving manner?*
- The core components of SPU include a frontend compiler and a backend MPC runtime.
- Diverse frameworks and libraries can be supported in SPU if there is a path from ML source code to PPHLO (Privacy-Preserving High-Level Operations).
- PPHLO enables us to propose and implement MPC-specific optimizations at both frontend and backend to achieve high performance.

1. Architecture Overview

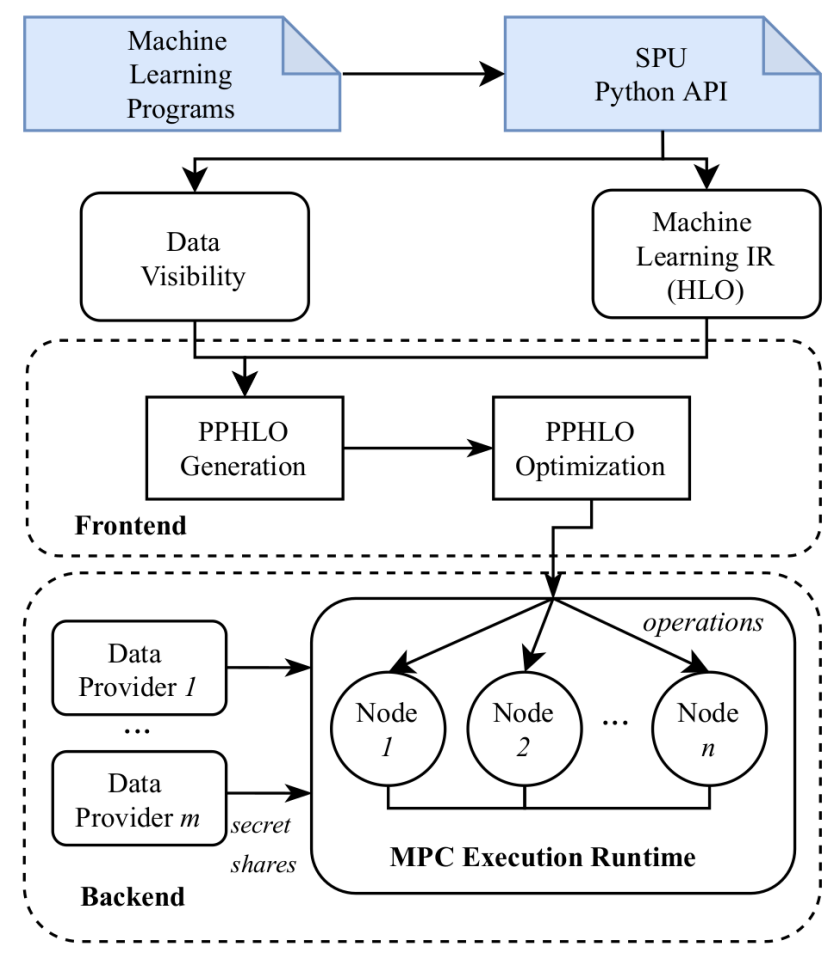

Figure 1: SPU architecture.

2. Privacy-Preserving High-Level Operations

- A tensor's type in PPHLO can be represented by a triple `<Shape, Data Type, Visibility>`
  - Visibility is a unique tensor attribute in PPHLO. It can be either secret or public.
- For each operation in PPHLO, we use the following rules to determine the output's type according to the input's type
  - **Data Type Promotion**: if one of the operands is a fixed-point number, the result is also a fixed-point number
  - **Visibility Narrowing**: if one of the operands is a secret, the result is also a secret.

3. Frontend

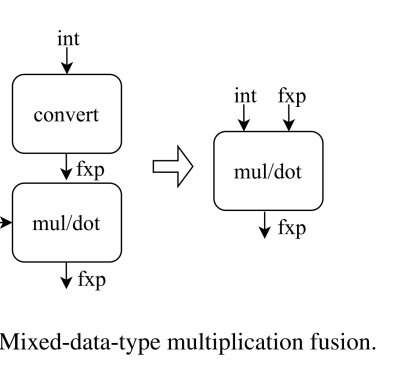

Figure 4: Mixed-data-type multiplication fusion.

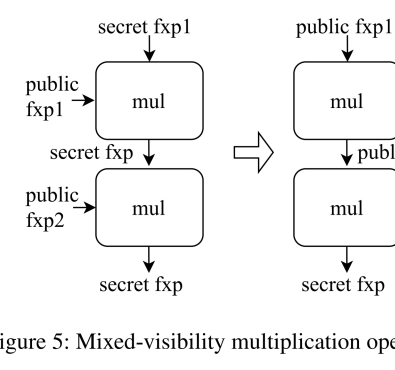

Figure 5: Mixed-visibility multiplication operands reorder.

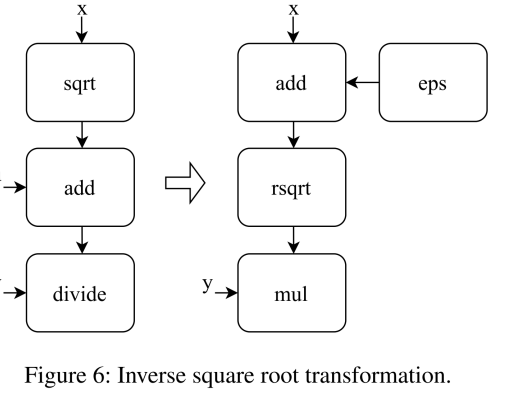

Figure 6: Inverse square root transformation.

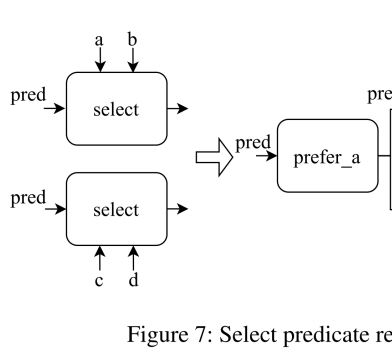

Figure 7: Select predicate reuse.

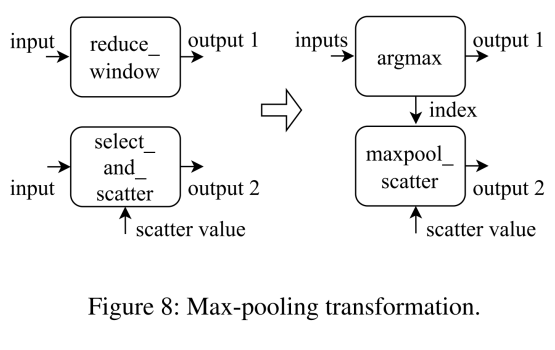

Figure 8: Max-pooling transformation.

4. Backend

Each MPC-primitive function will be finally dispatched to the MPC layer, corresponding to a specific implementation of fundamental MPC protocols. Adding a new MPC protocol in SPU only needs to implement the MPC-primitive function set.

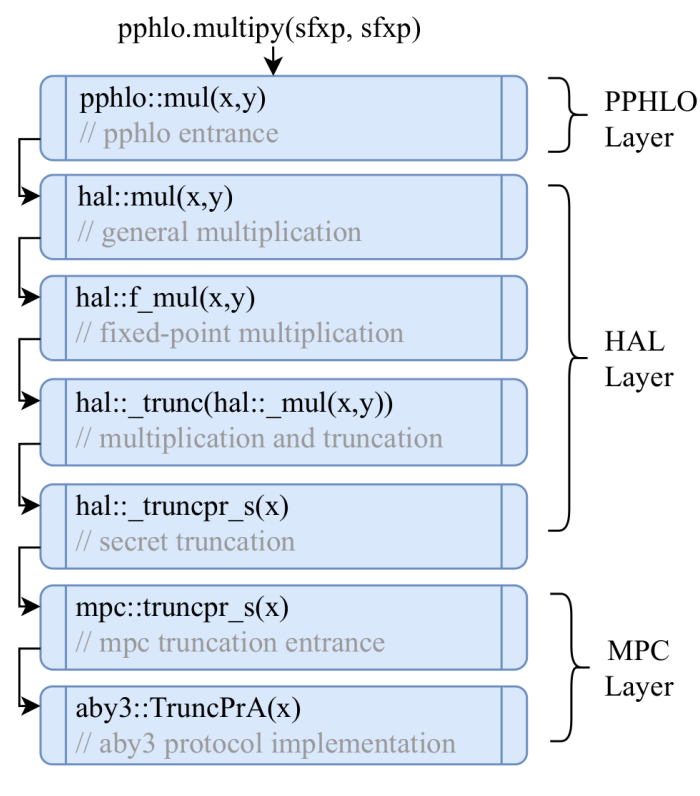

Figure 9: The dispatching path from a PPHLO operation to an MPC protocol in SPU. Different protocols can reuse the same PPHLO/HAL layer code and diverge at the final MPC layer.

Evaluation and Results

- Three state-of-the-art MPC-enabled PPML frameworks (i.e., MP-SPDZ, TF Encrypted, and CrypTen) as the baseline.
- We train four common-evaluated neural networks on the MNIST dataset for image classification under both local area network (LAN) and wide area network (WAN) settings.

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**

Notes

- The IR(intermediate representation) used in ML compilers is typically expressed as a computation graph (i.e., a directed acyclic graph).
- One widely-used ML compiler is Google's XLA. TensorFlow, PyTorch, and JAX, support XLA.
- One fundamental technique used in MPC is secret sharing.