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

Ant Group
USENIX ATC ’23, July 10, 2023
Machine Learning (ML) is powerful

Computer Vision
- ResNet, ViT

Natural Language Processing
- GPT, Bert, LLaMA

Drug Discovery
- AlphaFold, FastFold
Data usage in ML raises privacy concerns

Data is important
- Training high-quality ML models requires big-volume data
- Model services need users’ inputs for predictions

Data is sensitive
- Biometric data: images, voice, genome
- Financial data: income, expenses, liabilities
- Laws and regulations: GDPR
Data is important

- Training high-quality ML models requires big-volume data
- Model services need users’ inputs for predictions

Data is sensitive

- Biometric data: images, voice, genome
- Financial data: income, expenses, liabilities

Who Can Protect Your Data?
Solution: Secure Multiparty Computation (MPC)

Multiple parties jointly evaluate a function without leaking anything but the result.

3 parties compute an addition function
MPC enables Privacy-Preserving Machine Learning (PPML)

Private Training

Private Inference
Using MPC in PPML is challenging

MPC and ML worlds are naturally different

High-level building blocks

ML
- Forward/Backward computations
- SGD/Adam/AMSGrad Optimizers
- Tensors Operations
- CNN/Transformers/GNN SVM/K-means

MPC
- Semi-honest Malicious security
- Addition/Multiplication AND/XOR
- Secret Sharing Yao's garbled circuits
- Honest/Dishonest majority
- Mod Prime/$2^k$

Low-level cryptographic primitives
How do existing MPC-based PPML frameworks overcome this challenge?

Type I

General Purpose MPC Compilers
• Customized APIs
• Not compatible with ML frameworks

From Bottom to Top: Encapsulate cryptographic primitives into customized ML APIs
How do existing MPC-based PPML frameworks overcome this challenge?

Type I

General Purpose MPC Compilers
• Customized APIs
• Not compatible with ML frameworks

[CCS ’20]

A snippet from MP-SPDZ example

https://github.com/data61/MP-SPDZ/blob/master/Programs/Source/mnist_full_C.mpc
How do existing MPC-based PPML frameworks overcome this challenge?

Use ops provided in MP-SPDZ ML module

General Purpose MPC Compilers
- Customized APIs
- Not compatible with ML frameworks

```python
layers = [
    ml.FixConv2d([n_examples, 28, 28, 1], (20, 5, 5, 1), (20,), [N, 24, 24, 20], (1, 1), 'VALID'),
    ml.MaxPool([N, 24, 24, 20]),
    ml.ReLU([N, 12, 12, 20]),
    ml.FixConv2d(
        [N, 12, 12, 20], (50, 5, 5, 20), (50,), [N, 8, 8, 50], (1, 1), 'VALID'),
    ml.MaxPool([N, 8, 8, 50]),
    ml.ReLU([N, 4, 4, 50]),
    ml.Dense(N, 800, 500),
    ml.ReLU([N, 500]),
    ml.Dense(N, 500, 10),
]

optim = ml.Optimizer.from_args(program, layers)
optim.summary()
optim.run_by_args(program, n_epochs, batch_size, X, Y, acc_batch_size=N)
```

A snippet from MP-SPDZ’ example

https://github.com/data61/MP-SPDZ/blob/master/Programs/Source/mnist_full_C.mpc
How do existing MPC-based PPML frameworks overcome this challenge?

Use ops provided in MP-SPDZ ML module

- General Purpose MPC Compilers
  - Customized APIs
  - Not compatible with ML frameworks

Use MP-SPDZ supported optimizer

```python
layers = [
    ml.FixConv2d([n_examples, 28, 28, 1], (20, 5, 5, 1), (20,), [N, 24, 24, 20], (1, 1), 'VALID'),
    ml.MaxPool([N, 24, 24, 20]),
    ml.ReLU([N, 12, 12, 20]),
    ml.FixConv2d(
        [N, 12, 12, 20], (50, 5, 5, 20), (50,), [N, 8, 8, 50], (1, 1), 'VALID'),
    ml.MaxPool([N, 8, 8, 50]),
    ml.ReLU([N, 4, 4, 50]),
    ml.Dense(N, 800, 500),
    ml.ReLU([N, 500]),
    ml.Dense(N, 500, 10),
]

optim = ml.Optimizer.from_args(program, layers)
optim.summary()
optim.run_by_args(program, n_epochs, batch_size, X, Y,
                   acc_batch_size=N)
```

A snippet from MP-SPDZ example

https://github.com/data61/MP-SPDZ/blob/master/Programs/Source/mnist_full_C.mpc
How do existing MPC-based PPML frameworks overcome this challenge?

For complex programs like GPT-2 inference, users have to write them from scratch.
How do existing MPC-based PPML frameworks overcome this challenge?

Type II

TF/PyTorch-like Frameworks

• Offer TF/PyTorch-like APIs
• Looking like doesn't mean it is

[NeurIPS '21]

From Top to Bottom: Provide ML APIs with cryptographic implementations
How do existing MPC-based PPML frameworks overcome this challenge?

Type II

TF/PyTorch-like Frameworks
• Offer TF/PyTorch-like APIs
• Looking like doesn't mean it is

A snippet from CrypTen example

How do existing MPC-based PPML frameworks overcome this challenge?

TF/PyTorch-like Frameworks
• Offer TF/PyTorch-like APIs
• Looking like doesn't mean it is

Type II
torch tensor -> crypten tensor

# encrypt
x_alice_enc = crypten.cryptensor(x_alice, src=0)
x_bob_enc = crypten.cryptensor(x_bob, src=1)

# combine feature sets
x_combined_enc = crypten.cat([x_alice_enc,
                               x_bob_enc], dim=2)
x_combined_enc = x_combined_enc.unsqueeze(1)

# encrypt plaintext model
model_plaintext = CNN()
dummy_input = torch.empty((1, 1, 28, 28))
model = crypten.nn.from_pytorch(model_plaintext,
                                 dummy_input)
model.train()
model.encrypt()

A snippet from CrypTen example
How do existing MPC-based PPML frameworks overcome this challenge?

Type II

- torch tensor -> crypten tensor
- Offer TF/PyTorch-like APIs
- Looking like doesn't mean it is

A snippet from CrypTen example

How do existing MPC-based PPML frameworks overcome this challenge?

Type II

**torch tensor -> crypтен tensor**

TF/PyTorch-like Frameworks

- Offer TF/PyTorch-like APIs
- Looking like doesn't mean it is

**torch op -> crypтен op**

**torch model -> crypтен model**

# encrypt
x_alice_enc = crypтен.cryptensor(x_alice, src=0)
x_bob_enc = crypтен.cryptensor(x_bob, src=1)

# combine feature sets
x_combined_enc = crypтен.cat([x_alice_enc, x_bob_enc], dim=2)
x_combined_enc = x_combined_enc.unsqueeze(1)

# encrypt plaintext model
model_plaintext = CNN()
dummy_input = torch.empty((1, 1, 28, 28))

model = crypтен.nn.from_pytorch(model_plaintext, dummy_input)
model.train()
model.encrypt()

A snippet from Crypten example

How do existing MPC-based PPML frameworks overcome this challenge?

For complex ML programs like GPT-2 inference, users have to refactor TF/PyTorch programs by substituting supported PPML version APIs.

```python
# encrypt
x_alice_enc = crypten.cryptensor(x_alice, src=0)
x_bob_enc = crypten.cryptensor(x_bob, src=1)

# combine feature sets
x_combined_enc = crypten.cat([x_alice_enc, x_bob_enc], dim=2)

# encrypt plaintext model
model_plaintext = CNN()
dummy_input = torch.empty((1, 1, 28, 28))
model = crypten.nn.from_pytorch(model_plaintext, dummy_input)
model.train()
model.encrypt()
```
A question arises

Can we efficiently run ML programs of mainstream frameworks in a privacy-preserving manner?

```python
# encrypt
x_alice_enc = crypten.cryptensor(x_alice, src=0)
x_bob_enc = crypten.cryptensor(x_bob, src=1)

# combine feature sets
x_combined_enc = crypten.cat([x_alice_enc,
x_bob_enc], dim=2)
x_combined_enc = x_combined_enc.unsqueeze(1)

# encrypt plaintext model
model_plaintext = CNN()
dummy_input = torch.empty((1, 1, 28, 28))
model = crypten.nn.from_pytorch(model_plaintext, dummy_input)
model.train()
model.encrypt()
```
Our Answer: SecretFlow Secure Processing Unit (SPU)

Core Architecture Components
- Frontend: ML programs
- Compiler: Convert ML programs to PPHLO
- Runtime: Execute PPHLO as MPC protocols
Our Answer: SecretFlow Secure Processing Unit (SPU)

Core Architecture Components
- Frontend: ML programs
- Compiler: Convert ML programs to PPHLO
- Runtime: Execute PPHLO as MPC protocols
Our Answer: SecretFlow Secure Processing Unit (SPU)

Core Architecture Components

- Frontend: ML programs
- Compiler: Convert ML programs to PPHLO
- Runtime: Execute PPHLO as MPC protocols

![Diagram of Core Architecture Components]

- **Frontend**
  - JAX
  - TensorFlow
  - PyTorch
  - XLA HLO

- **Compiler**
  - MLIR
  - PPHLO
  - MPC-specific optimization

- **Runtime**
  - Executor
  - ABY3
  - Cheetah
  - SPDZ2k

**SPU Architecture**
Our Answer: SecretFlow Secure Processing Unit (SPU)

Core Architecture Components
- Frontend: ML programs
- Compiler: Convert ML programs to PPHLO
- Runtime: Execute PPHLO as MPC protocols
Our Answer: SecretFlow Secure Processing Unit (SPU)

Core Architecture Components

- **Frontend**: ML programs
- **Compiler**: Convert ML programs to PPHLO
- **Runtime**: Execute PPHLO as MPC protocols

![Diagram of SecretFlow architecture](image)
Our Answer: SecretFlow Secure Processing Unit (SPU)

Main Design Objectives
- Usability
- Extensibility
- High-performance
Our Answer: SecretFlow Secure Processing Unit (SPU)

Main Design Objectives
- Usability
- Extensibility
- High-performance

SPU bridges the gap
Usability: a GPT-2 example

Plaintext inference on CPU

```python
# greedy search
def text_generation(input_ids, params, token_num=10):
    config = GPT2Config()
    model = FlaxGPT2LMHeadModel(config=config)
    for _ in range(token_num):
        outputs = model(input_ids=input_ids, params=params)
        next_token_logits = outputs[0][0, -1, :]
        next_token = jnp.argmax(next_token_logits)
        input_ids = jnp.concatenate([[input_ids],
                                      jnp.array([[next_token]]),], axis=1)
    return input_ids

def run_on_cpu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog',
                                  return_tensors='jax')
    outputs_ids = text_generation(inputs_ids,
                                   pretrained_model.params)
    return outputs_ids
```

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2
SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
Usability: a GPT-2 example

Ciphertext inference on SPU

```python
# greedy search
def text_generation(input_ids, params, token_num=10):
    config = GPT2Config()
    model = FlaxGPT2LMHeadModel(config=config)
    for _ in range(token_num):
        outputs = model(input_ids=input_ids, params=params)
        next_token_logits = outputs[0][0, -1, :]
        next_token = jnp.argmax(next_token_logits)
        input_ids = jnp.concatenate([input_ids, jnp.array([[next_token]])], axis=1)
    return input_ids
def run_on_spu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog',
    return_tensors='jax')
    input_ids = ppd.device("P1")(lambda x: x)(inputs_ids)
    params = ppd.device("P2")(lambda x:
        x)(pretrained_model.params)
    outputs_ids = ppd.device("SPU")(text_generation,
        )(input_ids, params)
    outputs_ids = ppd.get(outputs_ids)
    return outputs_ids
```

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2
SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
Usability: a GPT-2 example

---

**CPU version**

```python
def run_on_cpu():
    inputs_ids = tokenizer.encode(
        'I enjoy walking with my cute dog',
        return_tensors='jax')
    outputs_ids = text_generation(inputs_ids, pretrained_model.params)
    return outputs_ids
```

**SPU version**

```python
def run_on_spu():
    inputs_ids = tokenizer.encode(
        'I enjoy walking with my cute dog',
        return_tensors='jax')

    input_ids = ppd.device("P1")(lambda x: x)(inputs_ids)
    params = ppd.device("P2")(lambda x: x)(pretrained_model.params)
    outputs_ids = ppd.device("SPU")(text_generation, )
                   (input_ids, params)
    outputs_ids = ppd.get(outputs_ids)
    return outputs_ids
```

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2

SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
Usability: a GPT-2 example

---

**CPU version**

```python
def run_on_cpu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='jax')
    outputs_ids = text_generation(inputs_ids, pretrained_model.params)
    return outputs_ids
```

**SPU version**

```python
def run_on_spu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='jax')
    input_ids = ppd.device("P1")(lambda x: x)(inputs_ids)
    params = ppd.device("P2")(lambda x: x)(pretrained_model.params)
    outputs_ids = ppd.device("SPU")(text_generation, )(input_ids, params)
    outputs_ids = ppd.get(outputs_ids)
    return outputs_ids
```

Diff

```python
input_ids = ppd.device("P1")\(\lambda x: x\)(inputs_ids)
params = ppd.device("P2")\(\lambda x: x\)(pretrained_model.params)
```

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2
SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
Usability: a GPT-2 example

<table>
<thead>
<tr>
<th>CPU version</th>
<th>SPU version</th>
</tr>
</thead>
<tbody>
<tr>
<td>def run_on_cpu():</td>
<td>def run_on_cpu():</td>
</tr>
<tr>
<td>inputs_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='jax')</td>
<td>tokenizer.encode('I enjoy walking with my cute dog', return_tensors='jax')</td>
</tr>
</tbody>
</table>
|     outputs_ids = text_generation(inputs_ids, pretrained_model.params) |     input_ids = ppd.device("P1")(lambda x: x)(inputs_ids)
|     return outputs_ids |     params = ppd.device("P2")(lambda x: x)(pretrained_model.params) |
|                |     outputs_ids = ppd.device("SPU")(text_generation, ) (input_ids, params) |
|                |     outputs_ids = ppd.get(outputs_ids) |
|                |     return outputs_ids |

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2
SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
def run_on_cpu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog',
        return_tensors='jax')

    outputs_ids = text_generation(inputs_ids, pretrained_model.params)
    return outputs_ids

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2

SPU version:
def run_on_spu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog',
        return_tensors='jax')

    input_ids = ppd.device("P1")(lambda x: x)(inputs_ids)
    params = ppd.device("P2")(lambda x: x)(pretrained_model.params)
    outputs_ids = ppd.device("SPU")(text_generation, )
        (input_ids, params)
    outputs_ids = ppd.get(outputs_ids)
    return outputs_ids

Load model.params at the party #2

SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
def run_on_cpu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='jax')
    outputs_ids = text_generation(inputs_ids, pretrained_model.params)
    return outputs_ids

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2

SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py

Usability: a GPT-2 example

Send $\text{input\_ids} \& \text{model\_params}$ to SPU for private inference
def run_on_cpu():
    inputs_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors='jax')
    outputs_ids = text_generation(inputs_ids, pretrained_model.params)
    return outputs_ids

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2

SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
Usability: a GPT-2 example

---

**ML ----> PPML**

**Modify several lines of code!**

**CPU version**

```python
def run_on_cpu():
    inputs_ids = tokenizer.encode(
        'I enjoy walking with my cute dog',
        return_tensors='jax'
    )
    outputs_ids = text_generation(inputs_ids, pretrained_model.params)
    return outputs_ids
```

**SPU version**

```python
def run_on_spu():
    inputs_ids = tokenizer.encode(
        'I enjoy walking with my cute dog',
        return_tensors='jax'
    )
    input_ids = ppd.device("P1")((lambda x: x)(inputs_ids))
    params = ppd.device("P2")((lambda x: x)(pretrained_model.params))
    outputs_ids = ppd.device("SPU")((text_generation, ))(input_ids, params)
    outputs_ids = ppd.get(outputs_ids)
    return outputs_ids
```

Adapted from the Huggingface GPT-2 Example: https://huggingface.co/docs/transformers/main/en/model_doc/gpt2

SPU version: https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_gpt2/flax_gpt2.py
Extensibility

Feasible to support multiple ML frameworks

If there is a path to XLA HLO, then there is a path to SPU
Extensibility

Easy to support multiple MPC protocols

Switch protocols by configurations

Reuse most code, adding protocols only needs implement a set of APIs
Performance: compiler

MPC-Specific DAG transformation

Mixed-visibility multiplication operands reorder

Max-pooling transformation
Performance: runtime

Efficient engineering implementation

Before tensor tiling

<table>
<thead>
<tr>
<th>Network I/O</th>
<th>Local Compute</th>
<th>Network I/O</th>
<th>Local Compute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After tensor tiling

<table>
<thead>
<tr>
<th>Network I/O</th>
<th>Local Compute</th>
<th>Network I/O</th>
<th>Local Compute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network I/O</td>
<td>Local Compute</td>
<td>Network I/O</td>
<td>Local Compute</td>
</tr>
</tbody>
</table>

Performance Improvement
Performance: evaluation

Training four neural networks under the semi-honest 3PC protocol

SPU’s Results

- Comparable accuracy
- Faster than SOTA for almost all settings
- Up to 4.1X faster than MP-SPDZ and up to 2.3X faster than TF Encrypted under the WAN setting

<table>
<thead>
<tr>
<th>Network</th>
<th>Network</th>
<th>Accuracy</th>
<th>Seconds per Batch (LAN)</th>
<th>Seconds per Batch (WAN)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>M</td>
<td>T</td>
</tr>
<tr>
<td>A (SGD)</td>
<td></td>
<td>96.8%</td>
<td>96.4%</td>
<td>92.7%</td>
</tr>
<tr>
<td>A (Adam)</td>
<td></td>
<td>97.5%</td>
<td>97.2%</td>
<td>N/A</td>
</tr>
<tr>
<td>A (AMSGrad)</td>
<td></td>
<td>97.6%</td>
<td>97.4%</td>
<td>N/A</td>
</tr>
<tr>
<td>B (SGD)</td>
<td></td>
<td>98.1%</td>
<td>98.3%</td>
<td>96.5%</td>
</tr>
<tr>
<td>B (Adam)</td>
<td></td>
<td>97.9%</td>
<td>98.7%</td>
<td>N/A</td>
</tr>
<tr>
<td>B (AMSGrad)</td>
<td></td>
<td>98.7%</td>
<td>98.8%</td>
<td>N/A</td>
</tr>
<tr>
<td>C (SGD)</td>
<td></td>
<td>98.5%</td>
<td>98.9%</td>
<td>97.3%</td>
</tr>
<tr>
<td>C (Adam)</td>
<td></td>
<td>98.8%</td>
<td>99.0%</td>
<td>N/A</td>
</tr>
<tr>
<td>C (AMSGrad)</td>
<td></td>
<td>99.2%</td>
<td>98.9%</td>
<td>N/A</td>
</tr>
<tr>
<td>D (SGD)</td>
<td></td>
<td>97.0%</td>
<td>97.6%</td>
<td>95.7%</td>
</tr>
<tr>
<td>D (Adam)</td>
<td></td>
<td>97.8%</td>
<td>98.0%</td>
<td>N/A</td>
</tr>
<tr>
<td>D (AMSGrad)</td>
<td></td>
<td>98.3%</td>
<td>97.5%</td>
<td>N/A</td>
</tr>
</tbody>
</table>

M: MP-SPDZ, T: TF Encrypted, C: CrypTen, S: SPU

Please refer to our paper for more details
THANKS!

Q & A

All code is available at: https://github.com/secretflow/spu

Issues are welcome for any questions!