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, Lei Wang

Ant Group

USENIX ATC '23, July 10, 2023

ANT GROUP

Machine Learning (ML) is powerful

Computer Vision

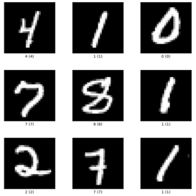
- ResNet, ViT

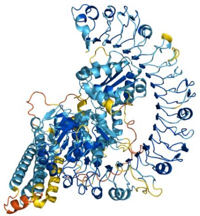
Natural Language Processing

- GPT, Bert, LLaMA

Drug Discovery

- AlphaFold, FastFold





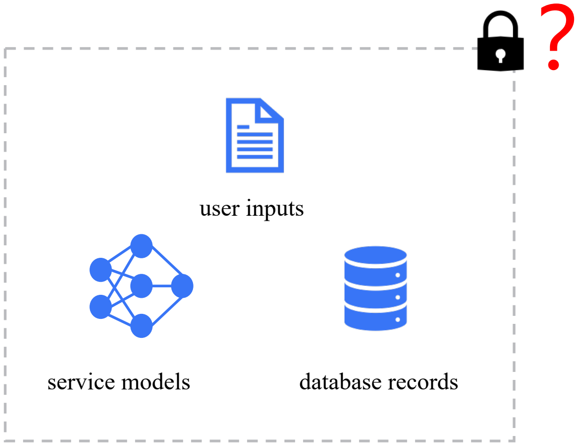
Data usage in ML raises privacy concerns

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



Who Can Protect Your Data?

Data is important

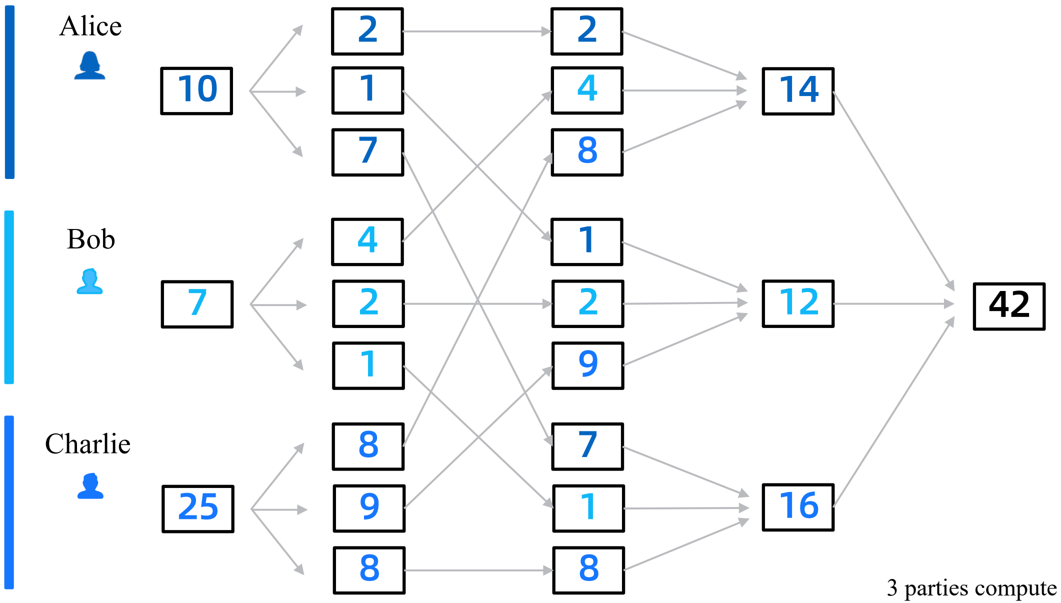
- Training high-quality ML models requires big-volume data
- Model services need users' inputs for predictions

Data is sensitive

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

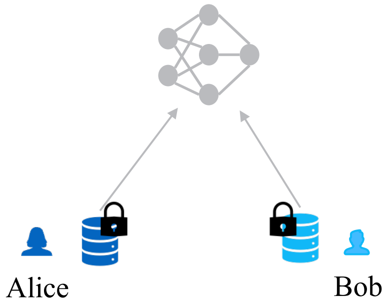
Solution: Secure Multiparty Computation (MPC)

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

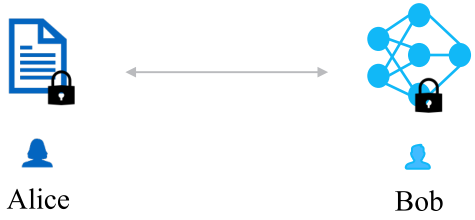


3 parties compute an addition function

MPC enables Privacy-Preserving Machine Learning (PPML)

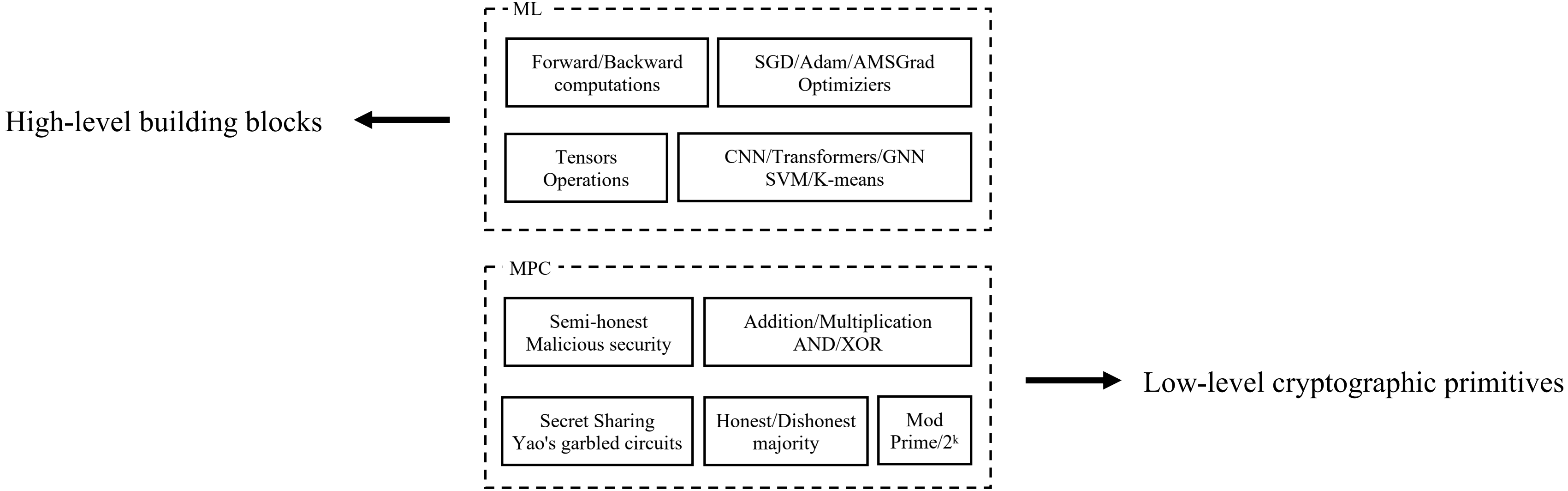


Private Training



Private Inference

Using MPC in PPML is challenging



MPC and ML worlds are naturally different

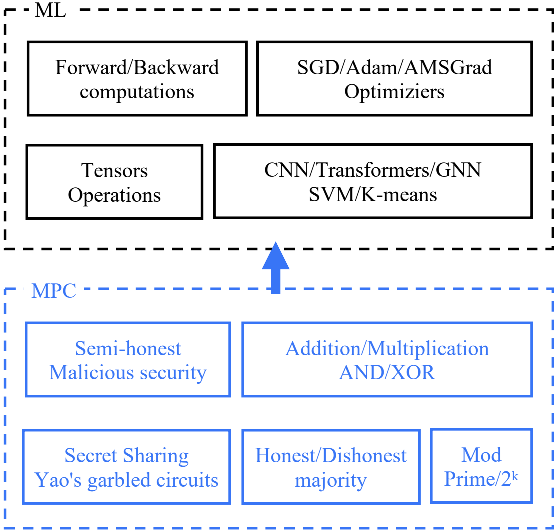
How do existing MPC-based PPML frameworks overcome this challenge?

Type I

General Purpose MPC Compilers

- Customized APIs
- Not compatible with ML frameworks

[CCS '20]



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

DATA 61 MP-SPDZ

[CCS '20]

```
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

Type I

General Purpose MPC Compilers

- Customized APIs
- Not compatible with ML frameworks

MP-SPDZ

[CCS '20]

```
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

Type I

General Purpose MPC Compilers

- Customized APIs
- Not compatible with ML frameworks

DATA61 MP-SPDZ

Use MP-SPDZ supported optimizer

[CCS '20]

```
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?

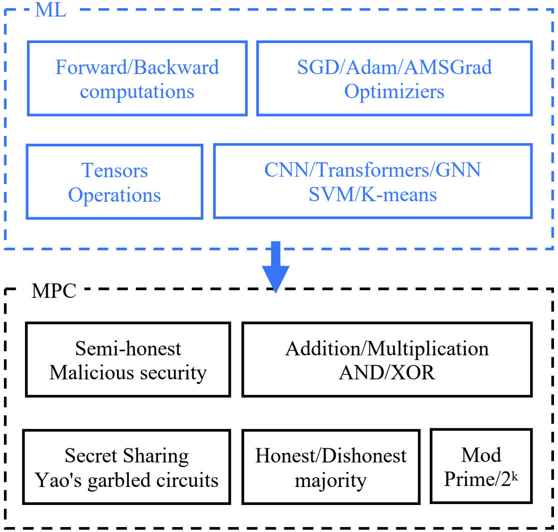
Type II

TF/PyTorch-like Frameworks

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

TFEncrypted CrypTen

[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

TFEncrypted CrypTen

[NeurIPS '21]

```
# 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

https://github.com/facebookresearch/CrypTen/blob/main/examples/mpc_autograd_cnn/mpc_autograd_cnn.py

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

Type II

torch tensor -> crypten tensor

TF/PyTorch-like Frameworks

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

TFEncrypted CrypTen

[NeurIPS '21]

```
# 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

https://github.com/facebookresearch/CrypTen/blob/main/examples/mpc_autograd_cnn/mpc_autograd_cnn.py

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

Type II

TF/PyTorch-like Frameworks

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

TFEncrypted CrypTen

[NeurIPS '21]

torch tensor -> crypten tensor

torch op -> crypten op

```
# 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

https://github.com/facebookresearch/CrypTen/blob/main/examples/mpc_autograd_cnn/mpc_autograd_cnn.py

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

torch tensor -> crypten tensor

torch op -> crypten op

torch model -> crypten model

```
# 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

https://github.com/facebookresearch/CrypTen/blob/main/examples/mpc_autograd_cnn/mpc_autograd_cnn.py

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

A question arises

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

Type II

TF/PyTorch-like Frameworks

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

TFEncrypted CrypTen

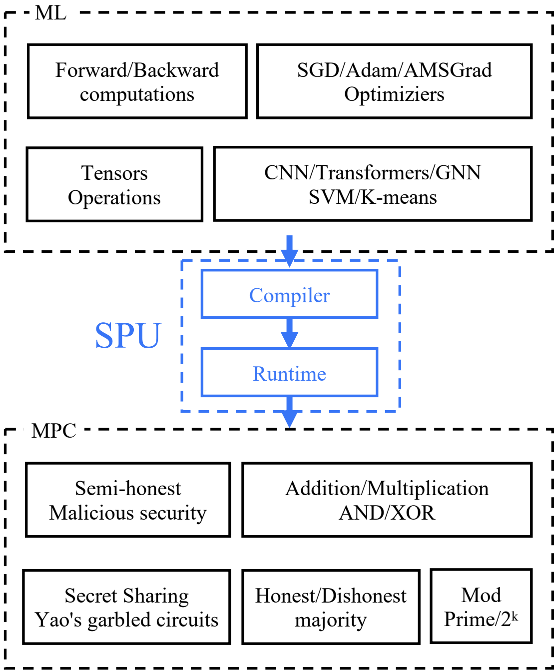
[NeurIPS '21]

```
# 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

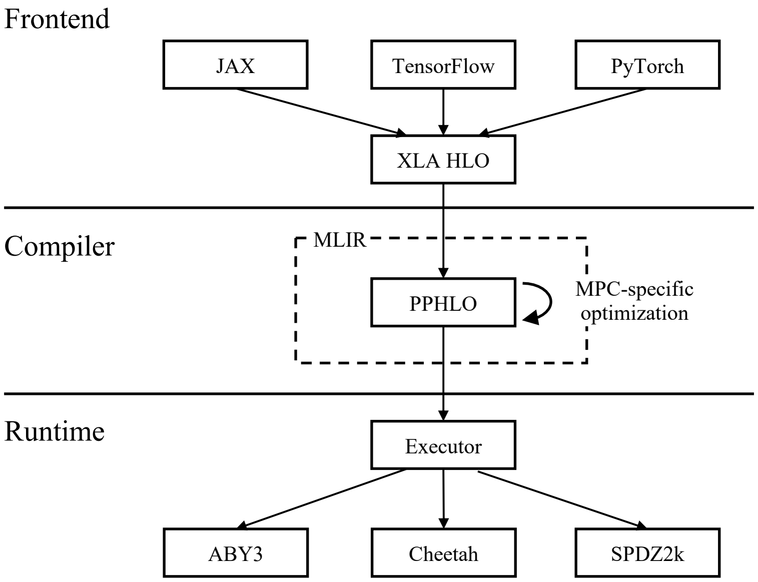
- Frontend: ML programs
- Compiler: Convert ML programs to PPHLO
- Runtime: Execute PPHLO as MPC protocols



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

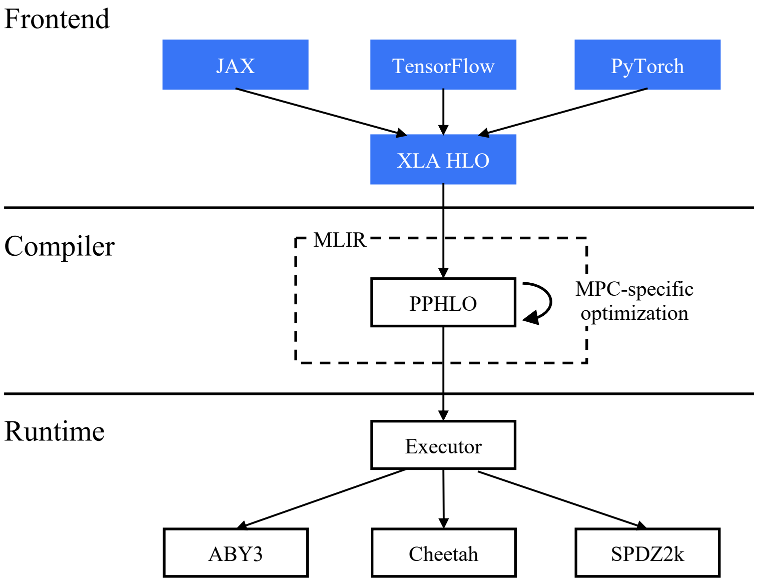


SPU Architecture

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



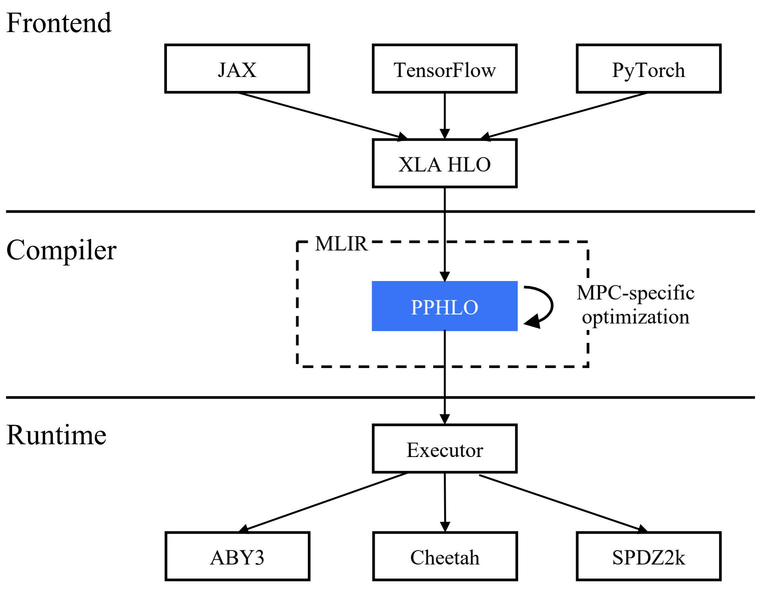
SPU Architecture

ANT GROUP SECRETFLOW

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

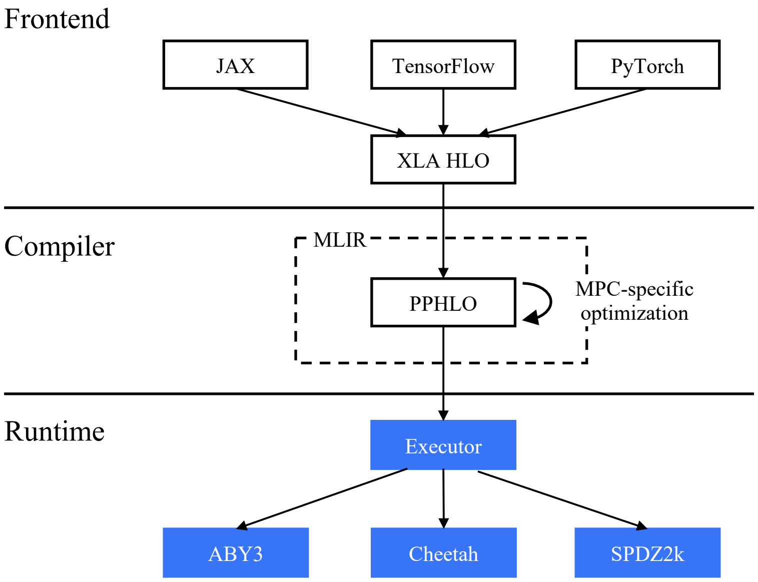


SPU Architecture

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

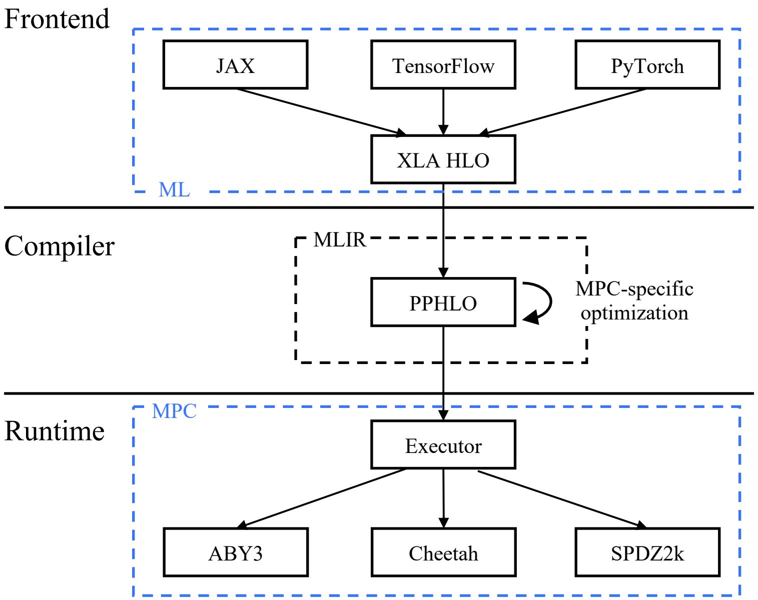


SPU Architecture

Our Answer: SecretFlow Secure Processing Unit (SPU)

Main Design Objectives

- Usability
- Extensibility
- High-performance



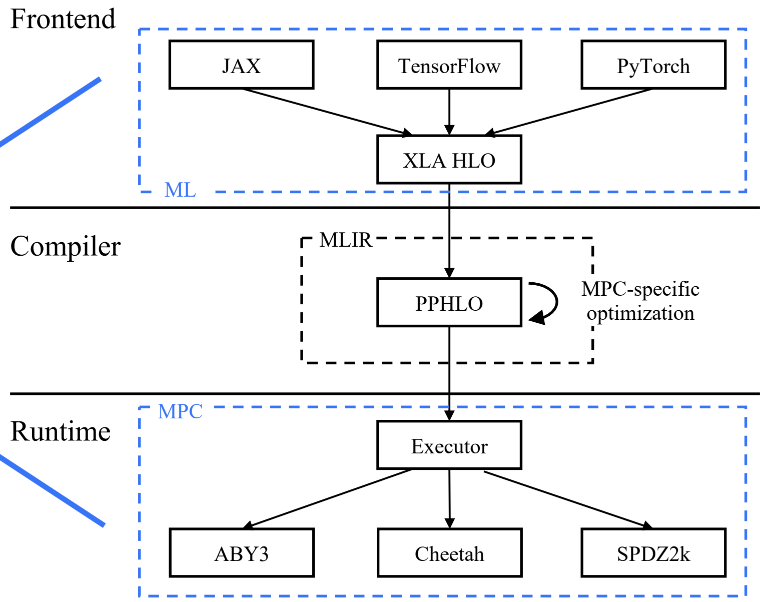
SPU Architecture

Our Answer: SecretFlow Secure Processing Unit (SPU)

Main Design Objectives

- Usability
- Extensibility
- High-performance

SPU bridges the gap



SPU Architecture

Usability: a GPT-2 example

Plaintext inference on CPU

```
# 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

```
# 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

```
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

```
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

SPU version

Diff

```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
```

```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

Load *input_ids* at the party #1

CPU version

```
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

```
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

Load *model.params* at the party #2

CPU version

```
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

```
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

Send *input_ids* & *model.params* to SPU for private inference

CPU version

```
def run_on_cpu():
    inputs_
        'I enjoy walking with my cute dog',
        return_tensors='jax')

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

SPU version

```
def run_on_spu():
    
        '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

Reveal the final *outputs_ids*

CPU version

```
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

```
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

ML ----> PPML

Modify several lines of code!

CPU version

```
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

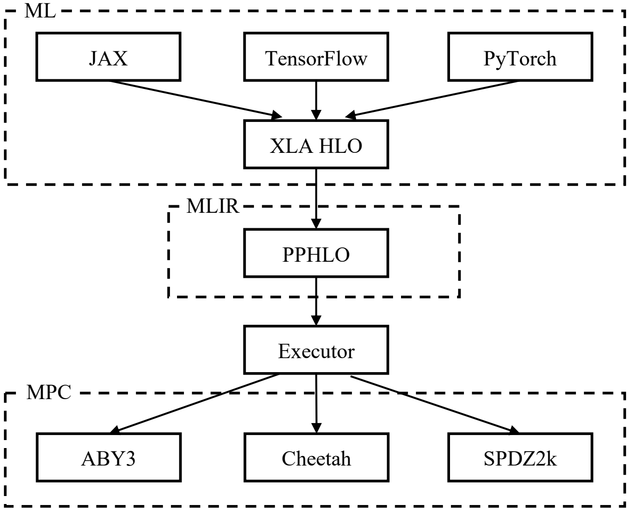
```
def run_on_spu():
    inputs_ids = tokenizer.encode(
                    'I enjoy walking with my cute dog',
                    return_tensors='jax')

    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



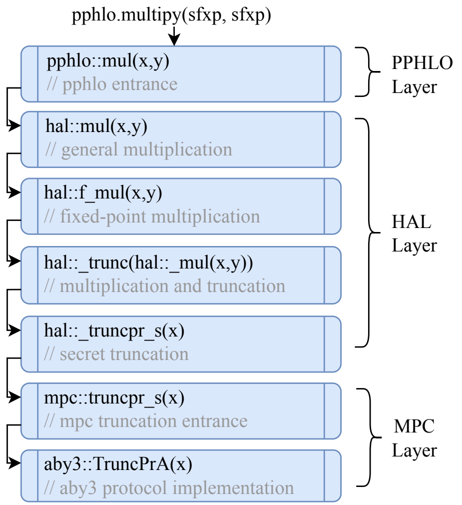
If there is a path to XLA HLO, then there is a path to SPU

Extensibility

Easy to support multiple MPC protocols

```
"SPU": {
  "kind": "SPU",
  "config": {
      "node_ids": ["node:0", "node:1", "node:2"],
      "runtime_config": {
          "protocol": "ABY3",
          "field": "FM64"
      }
  }
},

"SPU": {
  "kind": "SPU",
  "config": {
      "node_ids": ["node:0", "node:1"],
      "runtime_config": {
          "protocol": "CHEETAH",
          "field": "FM64"
      }
  }
}
```

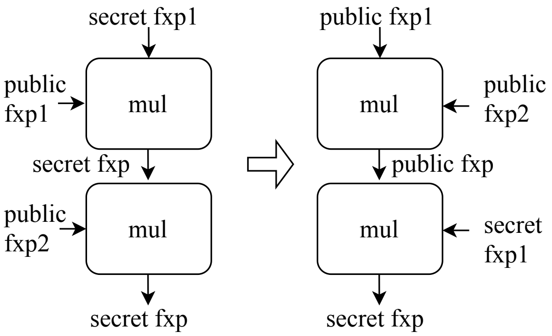
Switch protocols by configurations



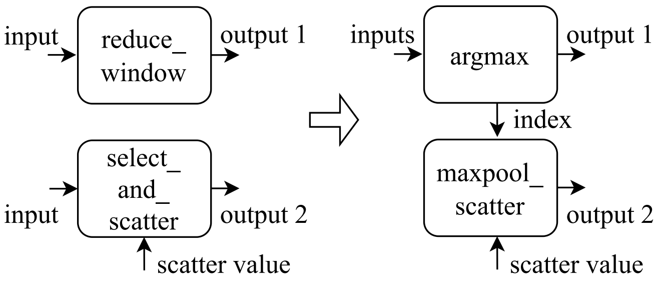
Reuse most code, adding protocols only needs implement a set of APIs

Performance: compiler

MPC-Specific DAG transformation



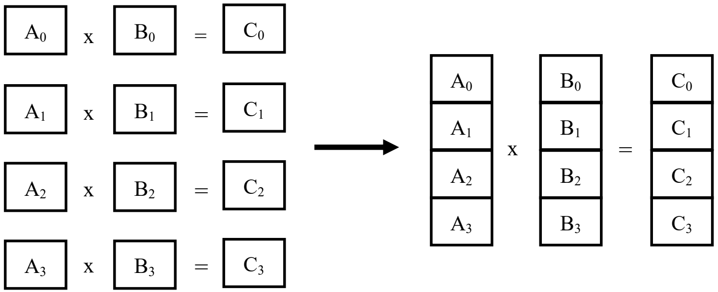
Mixed-visibility multiplication operands reorder



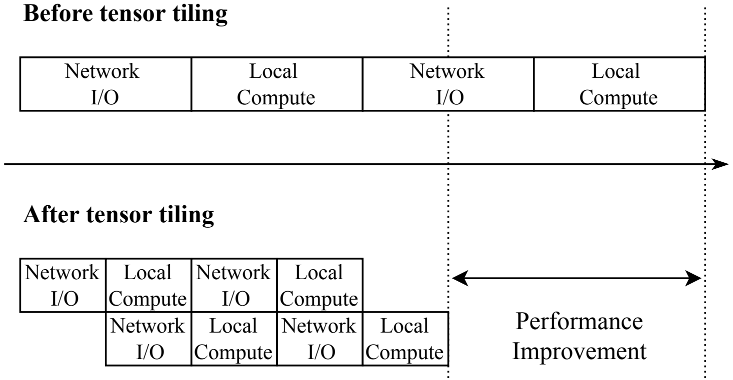
Max-pooling transformation

Performance: runtime

Efficient engineering implementation



Vectorization



Streaming

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

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**

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!

ANT GROUP