Cheetah
Lean and Fast Secure Two-Party
Deep Neural Network Inference

Zhicong (Zico) Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding
Alibaba Group
01 Background

02 Linear Primitives

03 Non-Linear Primitives

04 Performance and Summary
Secure Neural Network Inference

- Simple images and models
  - MNIST (28x28 black/white): 3~5 layers

- Complex images and models
  - CIFAR-10 (32x32 rgb): 10+ layers
  - IMAGENET (224x224 rgb): ResNet50

- ResNet50: one of the most popular DNN models

- Secure two-party ResNet50 inference
  - Prior best work: CryptFLOW2
  - 10 mins for one image inference (LAN, 3Gbps)
  - 20 mins for one image inference (WAN, 300Mbps)
### Design Challenges in 2PC Frameworks

- Optimize trade-offs among different primitives
- Adapt to concrete application cases

<table>
<thead>
<tr>
<th>Framework Type</th>
<th>Computation Cost</th>
<th>Communication Amount</th>
<th>Communication Round</th>
<th>Existing Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC (Y)</td>
<td>☆</td>
<td>☆☆☆</td>
<td>☆</td>
<td>EMP</td>
</tr>
<tr>
<td>SS (A, B)</td>
<td>☆</td>
<td>☆</td>
<td>☆☆☆</td>
<td>SPDZ, CryptFlow2</td>
</tr>
<tr>
<td>FHE</td>
<td>☆☆☆</td>
<td>☆</td>
<td>☆</td>
<td>Pegasus</td>
</tr>
<tr>
<td>A + B + Y</td>
<td>☆</td>
<td>☆☆☆</td>
<td>☆☆</td>
<td>ABY, SecureML</td>
</tr>
<tr>
<td>SS (A, B)</td>
<td>☆</td>
<td>☆</td>
<td>☆☆☆</td>
<td>Cheetah</td>
</tr>
</tbody>
</table>

- Low
- Medium
- High
Cheetah Protocol Architecture

Diagram:
- Sigmoid
- ReLU
- MaxPool
- FC
- CONV
- Truncation
- Inner Product
- SEAL BFV
- Compare
- AND triple
- Multiplexer
- B2A
- Silent ROT
- Silent 1-of-N OT
- Silent COT
- Silent GOT
- Silent Random Correlated OT
Additive Secret Sharing

• Integer $a \in [0, P)$ is split into shares $a_1, a_2$
  • Computation party $P_i$ has share $a_i$
  • Satisfy $a_1 + a_2 \mod P = a$

• Local Add/Sub computation

• Two types of sharings depending on modulus $P$
  • $P = 2 \rightarrow$ Boolean Share
  • $P > 2 \rightarrow$ Arithmetic Share. $P$ is usually a prime or a power of 2 (e.g., $2^{64}$)
02

Linear Primitives
Linear layers: CONV, FC

• CONV/FC: Matrix Mult $\rightarrow$ Inner Product

• Input:
  - Alice (model owner): vector $\vec{a}$
  - Bob (data owner): vector $\vec{b}$

• Output:
  - Alice: $r$
  - Bob: $\vec{a} \cdot \vec{b} - r \mod k$

![Diagram showing the interaction between Alice and Bob with Homomorphic Encryption]
Computation based on Polynomials

• Plaintext space for BFV: Polynomial Ring
  ➢ Polynomial $\mathbb{Z}_t(x)/x^{N+1}$
  ➢ Degree of N-1. Each integer coeff in [0, t-1]
  ➢ Ciphertext add/mult $\leftrightarrow$ Polynomial add/mult
  ➢ E.g.: $N = 2$, $t = 7 \rightarrow \text{mod } x^2 + 1$
    $\text{Enc}(x+2) \times \text{Enc}(x+3)$
    $= \text{Enc}(x^2+5x+6)$
    $= \text{Enc}(5x+5)$

A, B, C are polynomials
Packing: CRT Batching

• How to encode data into polynomials?
  ➢ \( x^n +1 \) can be broken into the product of \( n \) polynomials: \( x^n +1 = (x+a_1)(x+a_2)\ldots(x+a_n) \)
    • E.g.: \( t=17, n=2 \rightarrow x^2+1 = (x-4)(x-13) \)  // \( x^2-17x+52 \) mod 17
  ➢ \( f(x) \) mod \( (x^n +1) \) can represent \( n \) integers: \( x_i = f(x) \) mod \( (x+a_i) \)
    • E.g.: \( x \) mod \( (x^2+1) \rightarrow x \) mod \( (x-4) \) 和 \( x \) mod \( (x-13) \rightarrow x \) mod \( (x^2+1) \) “pack” 4 and 13

• Given \( n \) integers, find corresponding \( f(x) \) to encode them by CRT
  – E.g.: 2x-7 “pack” 1 and 2: // 2x-7 mod (x-4) = 1, 2x-7 mod (x-13) = 2 mod 17

• Packing keeps homomorphism modulo \( t \)
  – Add: \( x+(2x-7) \) packs 5 and 15: // 3x-7 mod (x-4) = 5, 3x-7 mod (x-13) = 15 mod 17
  – Mult: \( x(2x-7) \) packs 4 and 9:
    // 2x^2-7x mod (x^2+1) = -7x-2; -7x-2 mod (x-4) = 4, -7x-2 mod (x-13) = 9 mod 17

• SIMD: One polynomial calculation completes \( n \) integer calculations
Precondition of SIMD Packing in BFV

• Almost all efficient BFV applications use SIMD Packing
  ➢ One poly mult $\rightarrow$ 1000+ plain integer mults

• SIMD requires plain modulus $t$ to be a prime $\rightarrow$
  Secret sharing has to work in prime field in a mixed protocol
  ➢ Performance degrades significantly （60% more overhead [CrypTFlow2]）
Inner Product 1\textsuperscript{st} Try: SIMD Packing + Ciphertext Rotation

- A has a vector \(a=(a_0, a_1, \ldots, a_n)\), B has a vector \(b=(b_0, b_1, \ldots, b_n)\)
- A SIMD packs \(a\) as a polynomial \(A(x)/X^{N+1}\); B SIMD packs \(b\) as a polynomial \(B(x)/X^{N+1}\)
- B uses its public key to encrypt \(B(x)\), and send to A
- A performs homomorphic mult on \(\text{Enc}(B(x))\) and \(A(x)\) \(\to\) Obtains \(\text{Enc}(C(x))/X^{N+1}\)
  - \(C(x)\) packs \((a_0b_0, a_1b_1, \ldots, a_nb_n)\)
  - ! One step away from inner product: BIG SUM
- A \textbf{rotates} the ciphertext \(\text{Enc}(C(x))\), obtaining

\[
\begin{align*}
(a_1b_1, \ldots, a_{n-1}b_{n-1}, a_nb_n, a_0b_0) \\
(a_2b_2, \ldots, a_nb_n, a_0b_0, a_1b_1) \\
\vdots \\
(a_nb_n, a_0b_0, a_1b_1, \ldots, a_{n-1}b_{n-1})
\end{align*}
\]

\(\text{n ciphertexts}\)
- A performs homomorphic add to get \((a\cdot b, \ldots, a\cdot b)\), sends to B, and B decrypts to get \(a\cdot b\)
- ! Needs log\((n)\) rotates and \(n\) adds. Performance \textbf{not} better than Paillier.
Inner Product 2\textsuperscript{nd} Try: Polynomial Coefficient Encoding

- A has a vector \(a=(a_0,a_1, \ldots, a_n)\), B has a vector \(b=(b_0,b_1, \ldots, b_n)\)
  - A encodes \(a\) into a polynomial
    \[P_a = a_0 + a_1X + a_2X^2 + \cdots + a_nX^n\]
  - B encodes \(b\) into a polynomial
    \[P_b = b_0 - b_1X^{N-1} - b_2X^{N-2} - \cdots - b_nX^{N-n}\]
  - Where \(X^N = -1 \mod (X^N + 1)\)

- Hence the constant term of \(P_a*P_b\) is the inner product \(a \cdot b\)
- ✓ Only one homomorphic mult.
  - \(N=4096\) costs only 1 millisecond
2D Convolution

3×4

2×2

2×3

stride=1
2D Convolution

\[
\begin{array}{c}
\begin{array}{cccc}
    a_0 & a_1 & a_2 & a_3 \\
    a_4 & a_5 & a_6 & a_7 \\
    a_8 & a_9 & a_{10} & a_{11} \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{cc}
    b_0 & b_1 \\
    b_2 & b_3 \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{cc}
    \text{stride=1} \\
    \text{stride=2} \\
\end{array}
\end{array}
\end{array}
\]
2D Convolution

Encoding for Tensor

\[ a(X) = a_0 + a_1 X + a_2 X^2 + a_3 X^3 + a_4 X^4 + a_5 X^5 + a_6 X^6 + a_7 X^7 + a_8 X^8 + a_9 X^9 + a_{10} X^{10} + a_{11} X^{11} \]
2D Convolution

Encoding for Kernel

\[ a(X) = a_0 + a_1X + a_2X^2 + a_3X^3 + a_4X^4 + a_5X^5 + a_6X^6 + a_7X^7 + a_8X^8 + a_9X^9 + a_{10}X^{10} + a_{11}X^{11} \]

\[ b(X) = b_3 + b_2X + 0X^2 + 0X^3 + b_1X^4 + b_0X^5 \]

\[ a(X) \cdot b(X) = \sum_{i=0}^{15} c_iX^i \]
2D Convolution

Multiplication between a long polynomial and a short polynomial $\rightarrow$ Convolution

$$a(X) = a_0 + a_1X + a_2X^2 + a_3X^3 + a_4X^4 + a_5X^5 + a_6X^6 + a_7X^7 + a_8X^8 + a_9X^9 + a_{10}X^{10} + a_{11}X^{11}$$

$$b(X) = b_3 + b_2X + 0X^2 + 0X^3 + b_1X^4 + b_0X^5$$

$$a(X) \cdot b(X) = \sum_{i=0}^{15} c_iX^i$$

$$c_5 = a_0b_0 + a_1b_1 + a_4b_2 + a_5b_3$$
$$c_7 = a_2b_0 + a_3b_1 + a_6b_2 + a_7b_3$$
$$c_{10} = a_5b_0 + a_6b_1 + a_9b_2 + a_{10}b_3$$

$$c_6 = a_1b_0 + a_2b_1 + a_5b_2 + a_6b_3$$
$$c_9 = a_4b_0 + a_5b_1 + a_8b_2 + a_9b_3$$
$$c_{11} = a_6b_0 + a_7b_1 + a_9b_2 + a_{11}b_3$$
**Convolution**

**High flexibility: stride >= 1 & Same/Valid Padding & 3D Convolution**

\[ a(X) = a_0X^6 + a_1X^7 + \cdots + a_{11}X^{19} \]

\[ b(X) = b_3 + b_2X + 0X^2 + 0X^3 + 0X^4 + b_1X^5 + b_0X^6 \]

\[ a(X) \cdot b(X) = \sum_{i=0}^{25} c_iX^i \]
• The whole Tensor needs to be Encoded into a polynomial of degree $N$
  – $HW\cdot C \leq N$ (valid padding)
  – $(H - h + 1)(W - h + 1)C \leq N$ (same padding)
  – (rare case) when stride $s \geq h$, we can skip some computation
Big Tensor

Split along Channels

- The whole Tensor needs to be encoded into a polynomial of degree N
  - $HWC \leq N$ (valid padding)
  - $(H - h + 1)(W - h + 1)C \leq N$ (same padding)
  - (rare case) when stride $s \geq h$, we can skip some computation
- Big Tensor (e.g., $HWC > N$) can be split into small tensors
  - Along Channels: Just a simple addition in the end
The whole Tensor needs to be encoded into a polynomial of degree $N$

- $HWC \leq N$ (valid padding)
- $(H - h + 1)(W - h + 1)C \leq N$ (same padding)
- (rare case) when stride $s \geq h$, we can skip some computation

Big Tensor (e.g., $HWC > N$) can be split into small tensors

- Along Channels: Just a simple addition in the end
- Along Height/Width: Might contain overlaps
Multiple Kernels

Compute independently for each kernel

H x W x C kernels

- Same-padding
- Stride = s

M kernels

H/s x W/s x M
Non-Linear Primitives
Oblivious Transfer (Primitive)

• Sender has $\ell$-bit integers $a_0$, $a_1$
• Receiver chooses one of them with a choice bit $b \in \{0, 1\}$
• OT result:
  – Receiver gets $a_b$, but does not know $a_{1-b}$
  – Sender does not know $b$
• Other variants:
  – 1-of-$m$ OT: Sender has $m \geq 2$ messages
  – Random OT: Sender obtains random messages $a_0, a_1$
  – Correlated OT: Sender’s inputs $a_0, a_1$ satisfy some correlation (e.g., $a_1 = \Delta \oplus a_0$)
Non-linear layers (ReLU, MaxPool)

• ReLU: \( \text{ReLU}(x) := \max(x, 0) \)

• Input:
  - Alice, Bob: Secret-shared \( x \)

• Output:
  - Alice, Bob: Secret-shared \( \text{Compare}(x, 0) \times x \)

- \( \text{Compare}(x, 0) \)
  - \( = 0, \text{if } x < 0 \)
  - \( = 1, \text{if } x \geq 0 \)
Compare

- Compare\((b, a)\) solution 1: execute boolean adder to obtain MSB\((b - a)\)

\[
\begin{align*}
\text{a} &= 1110 \\
\text{b} &= 0100
\end{align*}
\]

Output secret-shared bits

\[
\begin{align*}
\text{Alice} & : 0011 \\
\text{Bob} & : 0001
\end{align*}
\]
• Solution 2: comparison tree [CrypTFlow2]
Compare

• Solution 2: comparison tree [CryptFlow2]

• Minimize comm. rounds and AND gates

Assume $x = a$

\[
\begin{align*}
&x < 0 \quad \text{0} \\
&x < 1 \\
&\ldots \\
&x < a \\
&x < a+1 \\
&\ldots \\
&x < 15 \\
\end{align*}
\]

1-of-16 OT

Alice inputs: $r \oplus \{x < i\}, \ 0 \leq i \leq 15$
Bob inputs: $y$

Alice obtains: $r$
Bob obtains: $r \oplus \{x < y\}$
Primitives in Compare

• 1-of-$2^m$ OT
• AND Gate
  - Beaver triple
  - 1-of-2 Random OT

• CryptFlow2 uses classic IKNP-OT
• Recent years, we have seen a series of Silent OT schemes based on VOLE
  - [CCS19], [Crypto21], [Ferret]
  - Generate massive amount of Random Correlated OT with little communication:
    \[ c_i = b_i + a_i \cdot x \]
    where \((b_i, b_i + x)\) are Sender’s random correlated messages, \(a_i \in \{0,1\}\) is Receiver’s choice bit
  - Random Correlated OT $\rightarrow$ Any other OT variant
Primitives in Compare

- 1-of-$2^m$ OT
  - IKNP-OT scheme: [KKOT 2013]
  - Silent OT scheme: $m$ instances of 1-of-2 Random OT [NaorPinkas1999]

Sender

$2^m$ messages: $x_0, x_1, ..., x_{2^m-1}$

Obtain: $(p_0^0, p_1^0), (p_0^1, p_1^1), ..., (p_0^{m-1}, p_1^{m-1})$

For all $k \in [0, 2^m)$, compute and send:

$y_k = x_k \oplus p_k^0 \oplus p_k^1 \oplus ... \oplus p_k^{m-1}$

Invoke $m$ ROTs

Receiver

choice $c \in [0, 2^m)$

Obtain: $p_{c[0]}, p_{c[1]}, ..., p_{c[m-1]}$

Compute:

$x_c = y_c \oplus p_{c[0]} \oplus p_{c[1]} \oplus ... \oplus p_{c[m-1]}$
## Primitives in Compare

<table>
<thead>
<tr>
<th>Primitives</th>
<th>Communication (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IKNP (CF2)</td>
</tr>
<tr>
<td>$2\choose 1$ - ROT$_{\ell}$</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>$2\choose 1$ - COT$_{\ell}$</td>
<td>$\ell + \lambda$</td>
</tr>
<tr>
<td>$2\choose 1$ - OT$_{\ell}$</td>
<td>$2\ell + \lambda$</td>
</tr>
<tr>
<td>$n\choose 1$ - OT$_{\ell}$ (n ≥ 3)</td>
<td>$n\ell + 2\lambda$</td>
</tr>
</tbody>
</table>

E.g.: $\ell = 64$, $\lambda = 128$
Truncation

• Fixed point (FP) numbers for MPC
  ➢ Value is 0.5, scale is $2^{15}$ → FP representation: $0.5 \times 2^{15} = 16384$

• Problem: multiplication increases the scale
  ➢ $0.5 \times 0.5 \rightarrow 16384 \times 16384 = 268435456 = 0.25 \times 2^{30}$
  ➢ Several mults would lead to an overflow

• Need a method to truncate secret-shared values to maintain the scale
  ➢ Plain truncation: $x >> 15$
  ➢ but local truncation leads to BIG error on secret sharings [SecureML]:
    $x = x_1 + x_2 \mod 2^k$
    $(x >> 15) \neq (x_1 >> 15) + (x_2 >> 15)$

  ➢ **Cheetah**: Efficient silent OT-based truncation protocol
    (1/2 probability with tiny one-bit LSB error)
04

Performance and Summary
## Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>System</th>
<th>End2End Time</th>
<th>Commu.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LAN</td>
<td>WAN</td>
</tr>
<tr>
<td>SqNet</td>
<td>SCI\textsubscript{HE} [50]</td>
<td>41.1s</td>
<td>147.2s</td>
</tr>
<tr>
<td></td>
<td>SecureQ8 [16]</td>
<td>4.4s</td>
<td>134.1s</td>
</tr>
<tr>
<td></td>
<td><strong>Cheetah</strong></td>
<td>16.0s</td>
<td>39.1s</td>
</tr>
<tr>
<td>RN50</td>
<td>SCI\textsubscript{HE} [50]</td>
<td>295.7s</td>
<td>759.1s</td>
</tr>
<tr>
<td></td>
<td>SecureQ8 [16]</td>
<td>32.6s</td>
<td>379.2s</td>
</tr>
<tr>
<td></td>
<td><strong>Cheetah</strong></td>
<td>80.3s</td>
<td>134.7s</td>
</tr>
<tr>
<td>DNet</td>
<td>SCI\textsubscript{HE} [50]</td>
<td>296.2s</td>
<td>929.0s</td>
</tr>
<tr>
<td></td>
<td>SecureQ8 [16]</td>
<td>22.5s</td>
<td>342.6s</td>
</tr>
<tr>
<td></td>
<td><strong>Cheetah</strong></td>
<td>79.3s</td>
<td>177.7s</td>
</tr>
</tbody>
</table>

SqNet = SqueezeNet; RN50 = ResNet50; DNet = DenseNet121

**SCI\textsubscript{HE}**: CryptFlow2  
**SecureQ8**: State-of-the-Art 3PC framework  

**Computation**: 3x  
**Communication**: 10x
### Takeaways

<table>
<thead>
<tr>
<th>Framework Type</th>
<th>Computation cost</th>
<th>Communication Amount</th>
<th>Communication Round</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS（A、B）</td>
<td>★</td>
<td>★</td>
<td>★★★</td>
<td>Cheetah</td>
</tr>
</tbody>
</table>

- With RLWE and Silent OT, 2PC systems can be implemented in very efficient ways
- The most optimized design need to consider computation tasks, primitives and parameters
  - Mod $2^k$ OR Mod $p$
  - Data encoding: SIMD OR Coefficient Encoding
  - Comparison: Adder circuit, Pure AND triple OR 1-of-N OT
  - ...

- Available:
  - [https://github.com/Alibaba-Gemini-Lab/OpenCheetah](https://github.com/Alibaba-Gemini-Lab/OpenCheetah)
THANKS