Goten: GPU-Outsourcing
Trusted Execution of Neural Network Training

Lucien K. L. Ng, Sherman S. M. Chow, Anna P. Y. Woo, Donald P. H. Wong (CUHK), and Yongjun Zhao (CUHK \(\rightarrow\) NTU)

Department of Information Engineering
Chinese University of Hong Kong (CUHK), Hong Kong
Training data for Neural Network

- **Sensitive**
  - Medical Image analysis, Child Exploitation Imagery, etc.
  - Privacy laws & Regulations, e.g., GDPR
- **Massive**
  - Hardly any single entity’s data is sufficient
- **Private Training**
  - No one should learn anything about the model & other’s data
Isn’t it solved by Federated Learning?

- **Federated Learning:**
  - Each data contributor train DNN locally
  - They exchange the DNN’s weight frequently

- **Problems:**
  - Every contributor can use the DNN
    - No rate-limiting, even for non-agreed/illegal uses
  - Contributors may steal others’ data
    - Model Inversion Attack (Fredrikson et al.)
  - Noisy/Implicit data $\Rightarrow$ Data privacy

Fredrikson, Somesh Jha, Thomas Ristenpart.
Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures.

xkcd.com/2169
State-of-the-Art Crypto Approach

- Falcon from PETS ’21
  😊 premier venue for research in Privacy-Enhancing Technologies
  😊 (our upcoming paper presents a better scheme for inference)
- Use non-colluding servers for private training
  😞 Take 5+ weeks to train VGG-16 for CIFAR-10

Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin.
Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning
H/W-assisted Approach: Slalom (ICLR’19)

- Slalom is assisted by hardware: Intel SGX & GPU
  😁 Free from heavyweight cryptographic tools
  😞 But it only supports private inference
    - Inference is easier than training (esp. for crypto-processing)
  😞 No model privacy
    - the server knows the plaintext model
    - no outsourcing of inference service

Florian Tramèr, Dan Boneh
Slalom: Fast, verifiable and private execution of neural networks in trusted hardware
TEE: Trusted Execution Environment

- e.g., Intel SGX

😊 protect the data’s privacy inside
  - even the machine owner cannot read it

😂 processes data efficiently as plaintext on CPU

😢 still works in CPU
  - too slow for batched linear operations
GPU: Graphics Processing Unit

😊 GPU can speed up the linear layers in DNNs
  - the linear layers is the most time-consuming part in DNNs

😊 GPU does not have TEE
  - lack of data privacy & model privacy!
Rundown

- GPU + TEE ➔ Private Training

- Slalom’s approach, and why it fails for training

- Goten
  - System Overview: Non-colluding servers
  - Core Technique: Additive secret sharing
  - (Dynamic) Quantization
  - Quick Discussion of Our Experimental Results
Rundown

- GPU + TEE → Private Training
- Slalom’s approach, and why it fails for training
- Goten
  - System Overview: Non-colluding servers
  - Core Technique: Additive secret sharing
  - (Dynamic) Quantization
  - Quick Discussion of Our Experimental Results
Slalom’s Approach

- Treat linear layers and non-linear layers differently
  - non-linear layers: e.g., ReLU, Maxpool
  - linear layers: e.g., Convolution, Matrix Multiplication
Slalom’s Linear Layers (and its two problems)

- Operation of a Linear Layer: \( y = W \otimes x \)
  - \( y \): output, \( x \): inputs, and \( W \): weight (e.g., kernel in a conv. layer)

**Offline Preparation** (Before the query arrives)

\[ y_r = W \otimes r \]

Randomly drawn

**Online Computation** (After the query arrives)

\[ x' = x - r \mod q \]

\[ y_u = W \otimes x' \]

\[ y = y_u + y_r \mod q \]

1\textsuperscript{st} Problem:
- \( \otimes \) stills run in (slow) CPU
  - Low Throughput
  - Fail to support secure outsourcing

2\textsuperscript{nd} Problem:
- The weight is exposed to the GPU
  - Fail to support private training
Contributors send their data to SGX’s TEE/enclaves
- Securely outsource linear-layer computation to GPUs
  - resided in with 3 non-colluding servers ($U_0$, $U_1$, $U_2$)
  - can reduce to 2 servers (at ½ of the throughput)
- Train (mostly) non-linear layers in SGX
Goten’s Training with GPU-Outsourcing

- **Init**: Training Data
- **Linear Layers**: Non-Linear Layers
- **Output Layer**: Back-propagate
- **Update Params**: 

Sync
- Training data, Randomness, NN architecture

Untrusted Zone

Goten
Non-Colluding Servers

- Each server holds a secret-share of the model/data
  - a setting adopted by many existing works (e.g., Falcon)
- Individual share by itself is totally random/meaningless
  - different from incomplete/partitioned data (e.g., federated learning)
- Candidates:
  - Some of the data contributors
    - More difficult to compromise different parties simultaneously
  - Government: Hospital/Monetary authority
    - If they are deemed trustworthy
  - Independent & Competing Cloud Server Providers
    - If they care their reputation
Non-colluding servers enable secure linear operation $\otimes$

- $U_0$ and $U_1$ hold the **additive secret shares** (SS) of $(W, x)$
  - $U_0$ holds $<W>_0$ and $<x>_0$
  - $U_1$ holds $<W>_1$ and $<x>_1$

- Homomorphism: linear $\otimes$ can be applied on $<W>_i$ and $<x>_i$
- Privacy: seeing $<W>_0$ (or $<W>_1$) learns nothing about $W$

- When computing $y = W \otimes x$, nothing exposes to the servers
- Now we can protect $W$! (vs. Slalom’s leakage of $W$ to the GPU)
Secret Sharing $x = <x>_0 + <x>_1 \pmod{q}$

- **Privacy ($<x>_i$ has no information about $x$):**
  - For each value of $x$, given $<x>_i$, there exists corresponding $<x>_{1-i}$

- **(Efficient) Homomorphic operation:**
  - $<x> + <y> = <x + y>$
  - For brevity, we will omit $(\text{mod } q)$

- **(Efficient) Generation:**
  - $\text{Gen}_i(r_x, x)$ generates $x_i$ for $i = 0, 1$
  - $\text{Rand}(r_x)$ picks $<x>_1$ uniformly at random from $\mathbb{Z}_q$
  - $<x>_0 = x - <x>_1$
  - Only 1 random number generation and 1 modular addition

- *(Easily generalizes to a matrix or a tensor)*
How Goten’s GPU-outsourcing works?

- **Goal:** Compute $y = W \otimes x$
- **Without leaking any** $(W, x, y)$ **to** $(U_0, U_1, U_2)$
- **High-Level Idea:**
  - $E_0$ and $E_1$ send secret secrets of $\{W, x\}$ to $U_0, U_1$
  - $U_0, U_1, U_2$ compute over the secret shares
  - $E_0$ and $E_1$ combine the “partial” results of $U_0, U_1, U_2$
How Goten’s GPU-outsourcing works?

- Goal: Compute \( y = W \otimes x \)
- Without leaking any \((W, x, y)\) to \((U_0, U_1, U_2)\)
- \(U_2\)’s bootstrapping:
  - \( u \leftarrow \text{Rand}(r_u), v \leftarrow \text{Rand}(r_v) \)
  - \( z \leftarrow u \otimes v \)

- \( \text{Gen}_0(r_x, x) \) and \( \text{Gen}_1(r_x, x) \) are generators for an additive SS of \( x \)
- \( \text{Rand}(r) \) is a secure pseudo-random generator
- \( \{r_u, r_v, r_x, r_W\} \) are pre-agreed random seeds (of the generators)
How Goten’s GPU-outsourcing works?

- **Goal:** Compute $y = W \otimes x$
- **Without leaking any $(W, x, y)$ to $(U_0, U_1, U_2)$**
- **$U_2$’s bootstrapping:**
  - $u \leftarrow \text{Rand}(r_u), v \leftarrow \text{Rand}(r_v)$
  - $z \leftarrow u \otimes v$
  - Send $z$ to $E_0$ and $E_1$

- $\text{Gen}_0(r_x, x)$ and $\text{Gen}_1(r_x, x)$ are generators for an additive SS of $x$
- $\text{Rand}(r)$ is a secure pseudo-random generator
- $\{r_u, r_v, r_x, r_W\}$ are pre-agreed random seeds (of the generators)
How Goten’s GPU-outsourcing works?

- **Goal:** Compute $y = W \otimes x$
- **Without leaking any** $(W, x, y)$ to $(U_0, U_1, U_2)$
- $E_i \in \{0, 1\}$ ’s Masking & Dispatch:
  - $<W>_i \leftarrow \text{Gen}_i(r_w, W)$
  - $<x>_i \leftarrow \text{Gen}_i(r_x, x)$
  - $e = W - \text{Rand}(r_u) = W - u$
  - $f = x - \text{Rand}(r_v) = x - v$

- $\text{Gen}_0(r_x, x)$ and $\text{Gen}_1(r_x, x)$ are generators for an additive SS of $x$
- $\text{Rand}(r)$ is a secure pseudo-random generator
- $\{r_u, r_v, r_x, r_w\}$ are pre-agreed random seeds (of the generators)
How Goten’s GPU-outsourcing works?

- **Goal:** Compute \( y = W \otimes x \)
- **Without leaking any \((W, x, y)\) to \((U_0, U_1, U_2)\)**
- \( E_i \in \{0, 1\} \)'s Masking & Dispatch:
  - \(<W>_i \leftarrow \text{Gen}_i(r_w, W)\)
  - \(<x>_i \leftarrow \text{Gen}_i(r_x, x)\)
  - \( e = W - \text{Rand}(r_u) = W - u \)
  - \( f = x - \text{Rand}(r_v) = x - v \)
  - Send \{\(<W>_i, <x>_i, e, f\)\} to \(U_i\)

- \( \text{Gen}_0(r_x, x) \) and \( \text{Gen}_1(r_x, x) \) are generators for an additive SS of \(x\)
- \( \text{Rand}(r) \) is a secure pseudo-random generator
- \{\(r_u, r_v, r_x, r_w\)\} are pre-agreed random seeds (of the generators)
How Goten’s GPU-outsourcing works?

- **Goal:** Compute \( y = W \otimes x \)
- **Without leaking any** \((W, x, y)\) **to** \((U_0, U_1, U_2)\)
- **Outsourced Computation:**
  - \( U_0 \) computes \( y_0 = <W>_0 \otimes f + e \otimes <x>_0 \)
  - \( U_1 \) computes \( y_1 = <W>_1 \otimes f + e \otimes <x>_1 - e \otimes f \)
How Goten’s GPU-outsourcing works?

- **Goal:** Compute $y = W \otimes x$
- **Without leaking any** $(W, x, y)$ **to** $(U_0, U_1, U_2)$

**Outsourced Computation:**

- $U_0$ computes $y_0 = <W>_0 \otimes f + e \otimes <x>_0$
- $U_1$ computes $y_1 = <W>_1 \otimes f + e \otimes <x>_1 - e \otimes f$
- $U_0$ sends $y_0$ to $E_0$ and $E_1$
- $U_1$ sends $y_1$ to $E_0$ and $E_1$

- $y_0 = <W>_0 \otimes f + e \otimes <x>$
- $y_1 = <W>_1 \otimes f + e \otimes <x>_1 - e \otimes f$
How Goten’s GPU-outsourcing works?

Goal: Compute $y = W \otimes x$

Without leaking any $(W, x, y)$ to $(U_0, U_1, U_2)$

$E_i \in \{0, 1\}$’s reconstruction of the results:

- $E_0$ and $E_1$ compute $y = y_0 + y_1 + z$

- $y_0 = <W>_0 \otimes f + e \otimes <x>$
- $y_1 = <W>_1 \otimes f + e \otimes <x>_1 - e \otimes f$
Correctness of the GPU Outsourcing

- $E_0$ and $E_1$ get $y$

- $y = y_0 + y_1 + z$
  
  \[
  y = y_0 + y_1 + z = \langle W \rangle_0 \otimes f + e \otimes <x>_0 + \langle W \rangle_1 \otimes f + e \otimes <x>_1 - e \otimes f + u \otimes v
  \]
  
  \[
  = W \otimes f + e \otimes x - e \otimes f + u \otimes v
  \]

- $e = W - u$

- $y_0 = \langle W \rangle_0 \otimes f + e \otimes <x>$
- $y_1 = \langle W \rangle_1 \otimes f + e \otimes <x>_1 - e \otimes f$

Diagram:

- $f = x - v$
- $e = W - u$
Security of the GPU Outsourcing

- What each non-colluding server sees:
  - \(U_0: <W>_0, <x>_0, e, f\)
  - \(U_1: <W>_1, <x>_1, e, f\)
  - \(U_2: u, v\)

- They are all secret shares or random tensors
  - \(<W>_{0/1}\) and \(<x>_{0/1}\) are secret shares (by definition)
  - \(e = (W - u)\) and \(f = (x - v)\) are secret shares
  - \(u\) and \(v\) are random tensors

- The security **only holds** over \(\mathbb{Z}_q\) (fixed-point integers)
  - Only then \(<W>_{0/1}, <x>_{0/1}, e, f\) are uniformly distributed (over their space)
Quantization for Secure Outsourcing

- (Linear layers’) Outsourcing protocol runs over $\mathbb{Z}_q$
  - $y_Q = W_Q \otimes_Q x_Q \mod q$ (as fixed points)
  - where $\otimes_Q$ denotes a linear operation over fixed points
- But non-linear layers work with floating points
  - They expect $y_f$ from linear layers
  - They output $x_f$ to linear layers
- We need (de)quantization!
  - $W_Q = \text{Quan}(W_f; \theta_W), x_Q = \text{Quan}(x_f; \theta_x)$
  - $y_f = \text{DeQ}(y_Q; \theta_W, \theta_x)$
Quantization for Secure Outsourcing

E₀ \( x_f, W_f \) → Quan → Mask → Unmask → DeQ → \( y_f \)

U₀

U₁

U₂

E₁ \( x_f, W_f \) → Quan → Mask → Unmask → DeQ → \( y_f \)
Yet another problem: Static Quantization

- Slalom (de)quantizes for cryptographic finite field:
  - Static Quantization: \( x_Q = \text{round}(x_f \cdot 2^8) \), \( w_Q = \text{round}(w_f \cdot 2^8) \)
  - Static Dequantization: \( y_f = \text{round}(y_Q \cdot 2^{-8}) \)
  - \( \theta_W \) and \( \theta_x \) are fixed to \( 2^8 \)
- The weight \((W)\) would fluctuate during training
  - A problem *explicitly mentioned* in Slalom’s paper!
  - When the entries of \( W_f \) becomes very small, \( y_Q \) becomes 0
    - \( w_Q = \text{round}(W_f \cdot 2^8) \approx 0 \rightarrow y_Q = W_Q \otimes_Q x_Q \approx 0 \)
  - When the entries \( W_f \) becomes very big, \( y_Q \) becomes trash
    - \( w_Q \approx \text{round}(W_f \cdot 2^8) > q \rightarrow \text{overflow} \)
    - Slalom sets \( q \approx 2^{24} \)
    - e.g., \( W_f \approx 2^{17} \rightarrow w_Q \approx \text{round}(2^{17} \cdot 2^8) \approx 2^{25} > q \)

Goten
Goten’s *Dynamic Quantization*

- The quantization param. \( (\theta_w, \theta_x) \) varies with the inputs
- Adapt from SWALP [Yang et al.]
  - a training scheme for low-bitwidth environment
- \( \theta_w \) is the magnitude of the maximum values of \( W \)
  - \( \theta_w = \lfloor \log_2 \cdot \max \cdot \text{abs}(W) \rfloor \)
  - \( \theta_x \) is also found similarly
- \( W_Q = \text{Quan}(W_f; \theta_w) = \text{clip}([W \cdot 2^{-\theta_w} 6], -2^7, 2^7) \)
  - \( x_Q \) is derived similarly
  - \( y_f = y_Q \cdot 2^{\theta_w + \theta_x - 12} \)

Guandao Yang, Tianyi Zhang, Polina Kirichenko, Junwen Bai, Andrew Gordon Wilson, Christopher De Sa.
SWALP: Stochastic Weight Averaging in Low-Precision Training.
ICML 2019
Dynamic Quantization’s Implications

- Accuracy drops <1%
  - from our experiments running with VGG-11 over CIFAR-10
- Dynamic quan. allows “proper utilization” of the bitwidth
  - The entries in $W_Q$ and $x_Q$ would not become 0 or overflow
How about Back-propagation?

- Back-prop. of linear layers are also linear operations
- e.g., fully-connected layers are all matrix multiplications
  - Forward: $y = x \times W^T$
  - Backward for $W$: $dW = dy^T \times x$
  - Backward for $x$: $dx = dy \times W$
Performance: Training over CIFAR-10

- Goten attains >89% accuracy in 34 hours
  - vs. Falcon’s 5 weeks (accuracy not reported)
- 132× throughput speed up over Falcon

<table>
<thead>
<tr>
<th>Framework</th>
<th>GPU</th>
<th>TEE</th>
<th>NN Architecture</th>
<th>Throughput</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Falcon</td>
<td>❌</td>
<td>❌</td>
<td>VGG-16</td>
<td>1482</td>
<td>132×</td>
</tr>
<tr>
<td>CaffeScone*</td>
<td>❌</td>
<td>✔</td>
<td>VGG-11</td>
<td>28800</td>
<td>6.84×</td>
</tr>
<tr>
<td>Goten</td>
<td>✔</td>
<td>✔</td>
<td>VGG-11</td>
<td>196733</td>
<td>-</td>
</tr>
</tbody>
</table>

- GPU: Nvidia V100 16GB
- CPU (w/ SGX): Intel i7-7700K
- Network: Google Cloud (8Gbps & <5ms latency)
- We run “hybrid” experiments due to resource constraints
  - More details in our paper

[*] Our pure-TEE private training framework over Caffe & SCONE (Secure Container Environment)
Invasive ductal carcinoma (IDC) detection

- Showcase application involving sensitive training data
- IDC: The most common type of breast cancer
- Dataset: Images of women’s breast tissue [Cruz-Roa et al.]

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>81%</th>
<th>82%</th>
<th>83%</th>
<th>84%</th>
<th>85%</th>
<th>86%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>8.53x</td>
<td>13.7x</td>
<td>4.27x</td>
<td>6.33x</td>
<td>3.42x</td>
<td>7.28x</td>
</tr>
<tr>
<td>Time (min)</td>
<td>1.25</td>
<td>1.56</td>
<td>13.1</td>
<td>16.9</td>
<td>31.2</td>
<td>46.8</td>
</tr>
</tbody>
</table>

Cruz-Roa et al.  
Medical Imaging: Digital Pathology 2014.
What we didn’t cover

- Large batch size is better for Goten
- Speedup ratio vs. Bandwidth
- CaffeScone (Our Pure-TEE Solution)’s optimal batch size
- Memory issues of SGX and our mitigation
- How to remove the third server
Closing Remarks

- A starting point for TEE + GPU private training
  - The Best of Both Worlds
- Our Techniques:
  - Lightweight Crypto for GPU-Outsourcing
  - Dynamic Quantization for Weight Fluctuation during Training
- Code: github.com/goten-team/Goten
- Our Another (Pure-Crypto) Secure Solution
  - "GPU-Friendly Oblivious and Rapid Classification Engine"
    - Conditionally Accepted by Usenix Security 2021
- Contact: {luciengkl, sherman}@ie.cuhk.edu.hk
Goten

- “Understand” (悟) the “sky” (天)

Image Credit: Dragon Ball