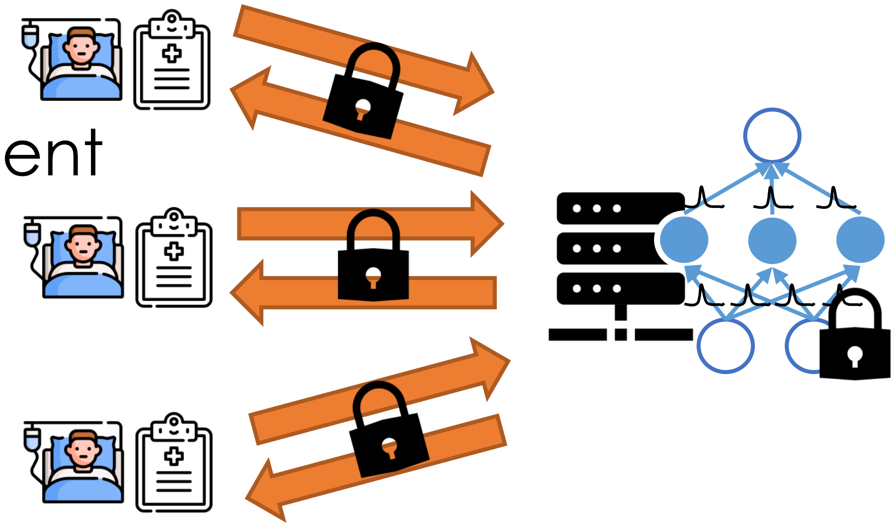
Goten: GPU-Outsourcing Trusted Execution of Neural Network Training

Lucien K. L. Ng, Sherman S. M. Chow, Anna P. Y. Woo,
Donald P. H. Wong (CUHK), and Yongjun Zhao (CUHK→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 $\nRightarrow$ Data privacy

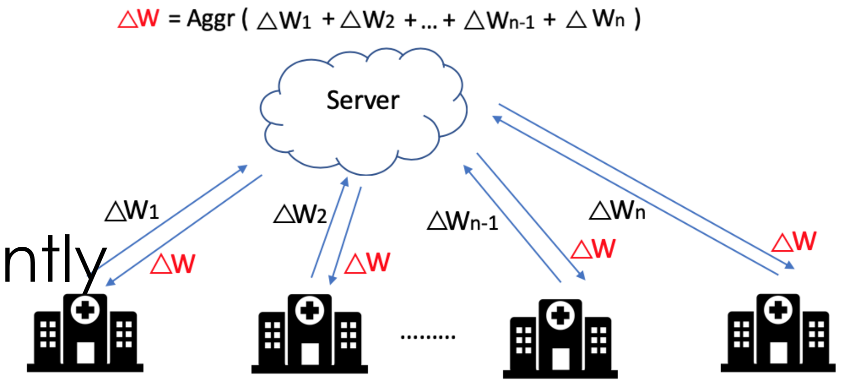


Image Credit: Vaikkunth Mugunthan

WHEN YOU TRAIN PREDICTIVE MODELS ON INPUT FROM YOUR USERS, IT CAN LEAK INFORMATION IN UNEXPECTED WAYS.

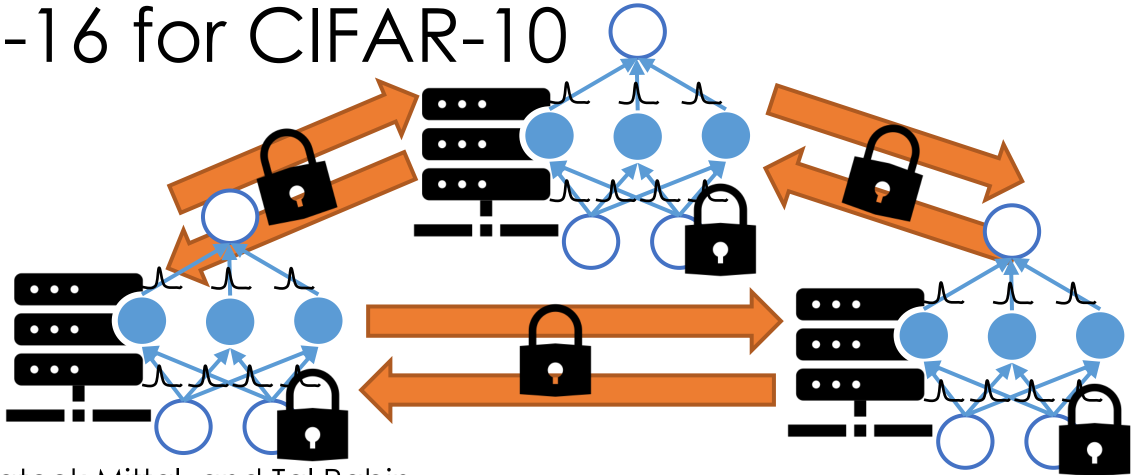
xkcd.com/2169

Matt Fredrikson, Somesh Jha, Thomas Ristenpart.
Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures.
ACM Conference on Computer and Communications Security (CCS) 2015.

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

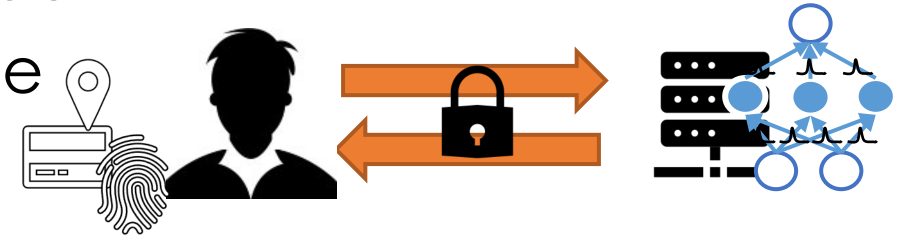
😁 Free from heavyweight cryptographic tools

☹️ 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



learn nothing about the ***model***

remains oblivious to ***query***

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

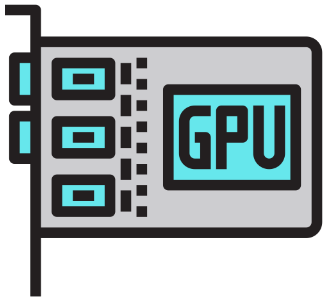
😁 processes data efficiently as plaintext on CPU

☹️ 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

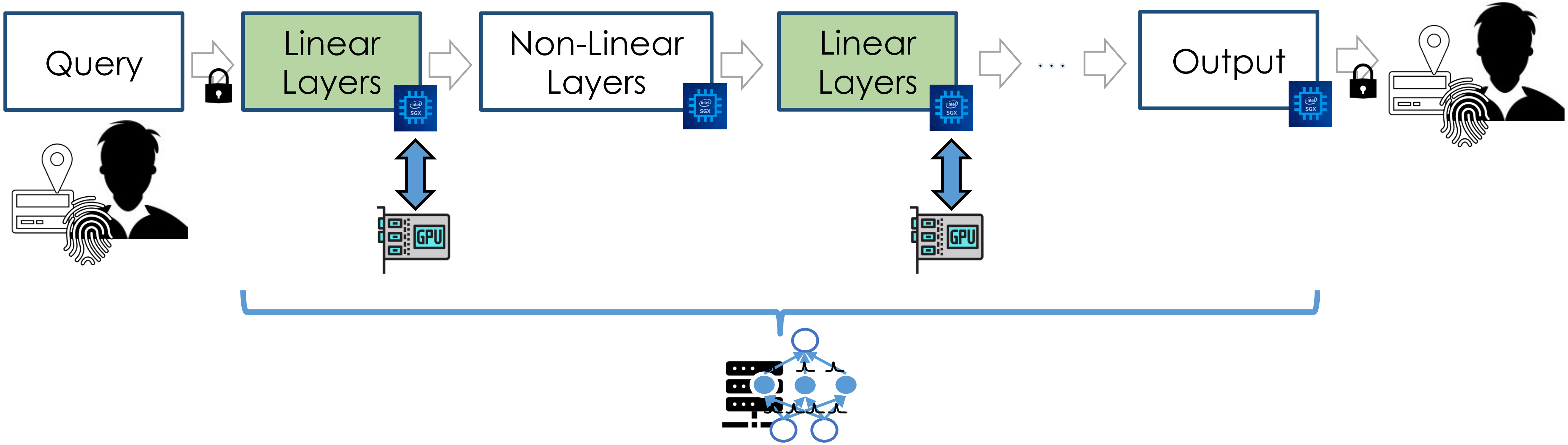
- Slalom's approach, and why it fails for training
- Goten
  - System Overview: Non-colluding servers
  - Core Technique: Additive secret sharing
  - (Dynamic) Quantization
  - Quick Discussion of Our Experimental Results

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

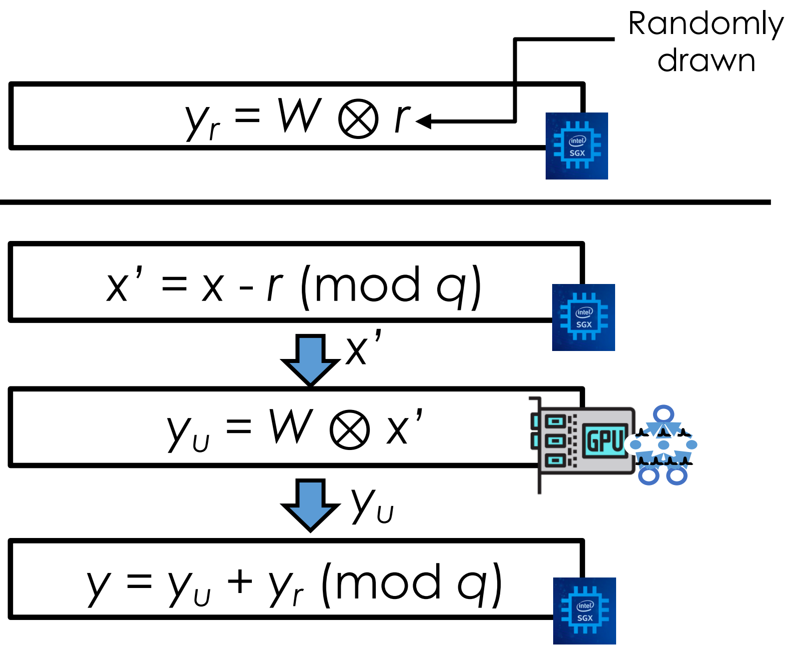


Slalom's Linear Layers (and its two problems)

- 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)

Online Computation
(After the query arrives)

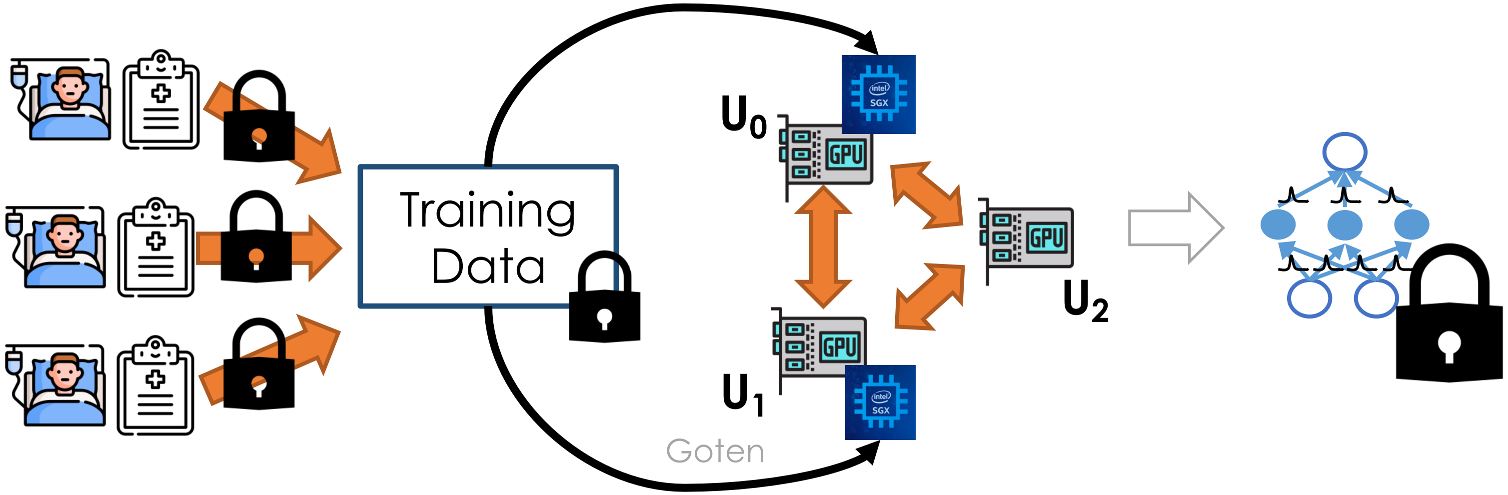


1st Problem:
$\otimes$ stills run in (slow) CPU
→ Low Throughput
(or lot of pre-computation & storage)

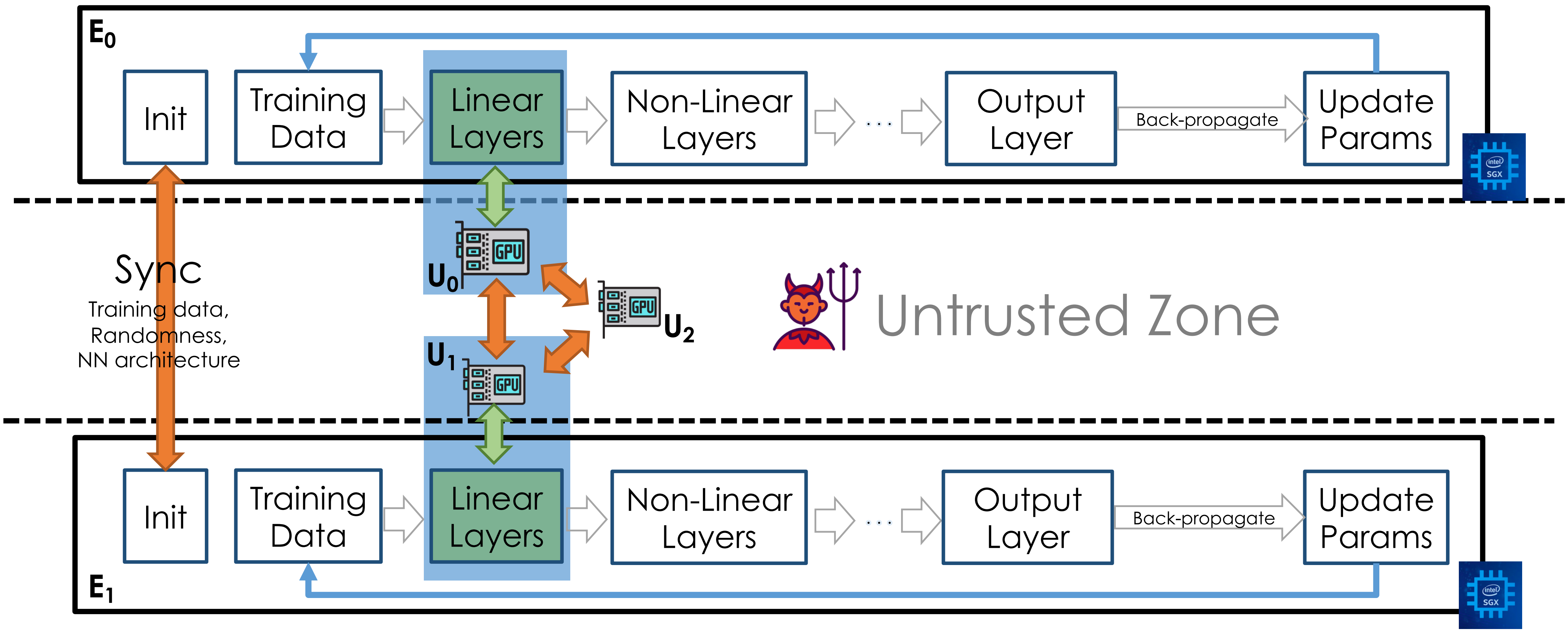
2nd Problem:
The weight is exposed to the GPU
→ Fail to support secure outsourcing
→ Fail to support private training

Goten: GPU + TEE for 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



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

Why Non-Colluding Servers?

- Non-colluding servers enable secure linear operation $\otimes$
- $\mathbf{U}_0$ and $\mathbf{U}_1$ hold the **additive secret shares** (*SS*) of (*W*, *x*)
  - $\mathbf{U}_0$ holds $<W>_0$ and $<x>_0$
  - $\mathbf{U}_1$ holds $<W>_1$ and $<x>_1$

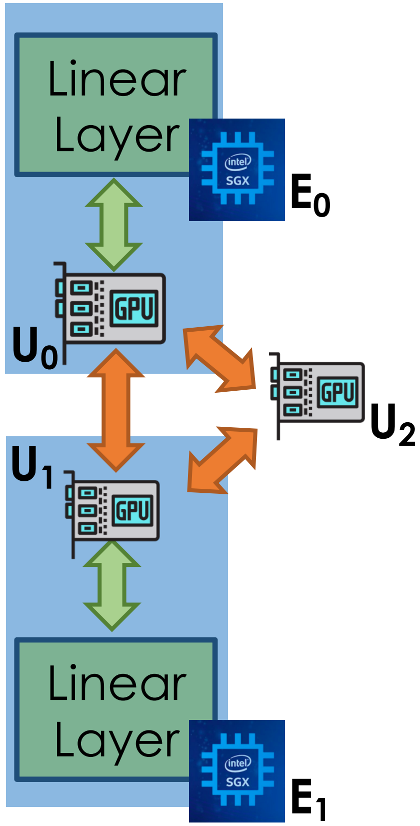
- Homomorphism: linear $\otimes$ can be applied on $<W>_i$ and $<x>_i$
- Privacy: seeing $<W>_0$ (or $<W>_1$) learns nothing about *W*

- 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 $x = \langle x \rangle_0 + \langle x \rangle_1 \pmod{q}$

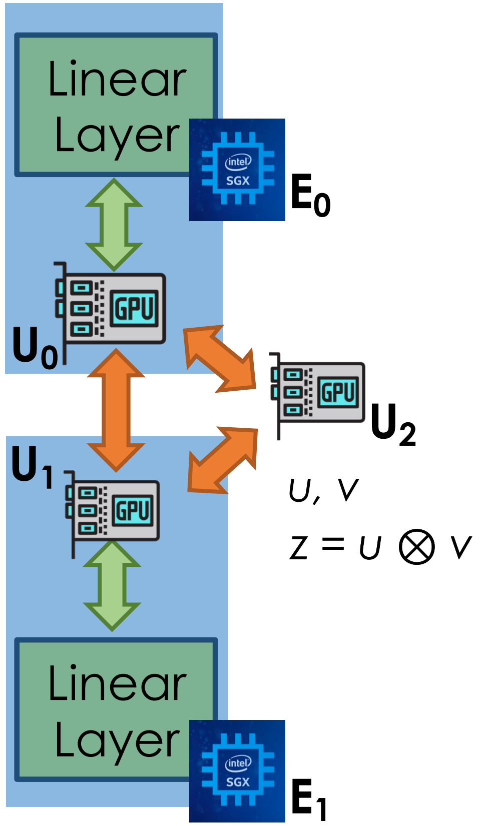
- Privacy ($\langle x \rangle_i$ has no information about x):
  - For each value of x, given $\langle x \rangle_i$, there exists corresponding $\langle x \rangle_{1-i}$
- (Efficient) Homomorphic operation:
  - $\langle x \rangle + \langle y \rangle = \langle x + y \rangle$
  - For brevity, we will omit (mod q)
- (Efficient) Generation:
  - $\mathrm{Gen}_i(r_x, x)$ generates x_i for $i = 0, 1$
  - $\mathrm{Rand}(r_x)$ picks $\langle x \rangle_1$ uniformly at random from $\mathbf{Z}_q$
  - $\langle x \rangle_0 = x - \langle x \rangle_1$
  - 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**, and **U_2** compute over the secret shares
  - **E_0** and **E_1** combine the "partial" results of **U_0**, **U_1**, **U_2**

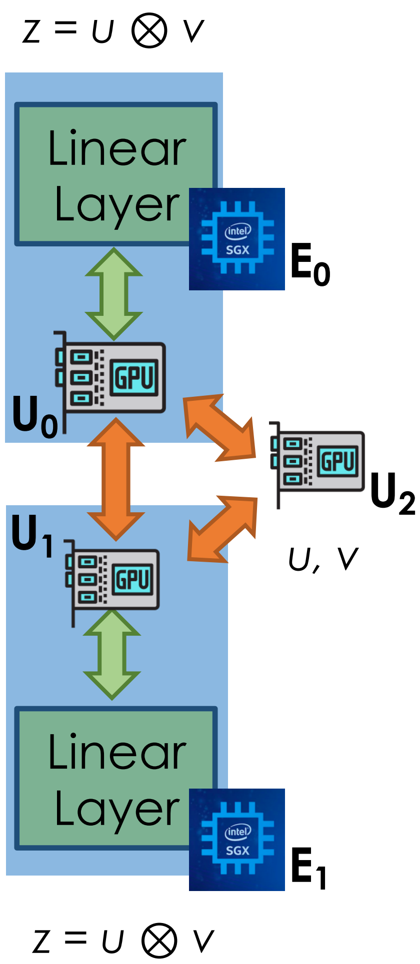
How Goten's GPU-outsourcing works?



- Goal: Compute $y = W \otimes x$
- Without leaking any (W, x, y) to $(\mathbf{U_0}, \mathbf{U_1}, \mathbf{U_2})$
- $\mathbf{U}_2$'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?



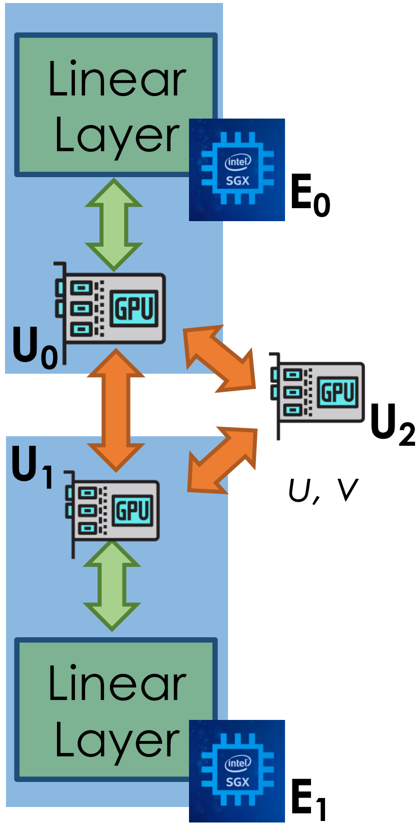
- Goal: Compute $y = W \otimes x$
- Without leaking any (W, x, y) to $(\mathbf{U_0}, \mathbf{U_1}, \mathbf{U_2})$
- $\mathbf{U}_2$'s bootstrapping:
  - $u \leftarrow \text{Rand}(r_u)$, $v \leftarrow \text{Rand}(r_v)$
  - $z \leftarrow u \otimes v$
  - Send z to $\mathbf{E_0}$ and $\mathbf{E_1}$

- $\text{Gen}_0(r_x, x)$ 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?

$z = u \otimes v$

$<x>_0, <W>_0, e, f$



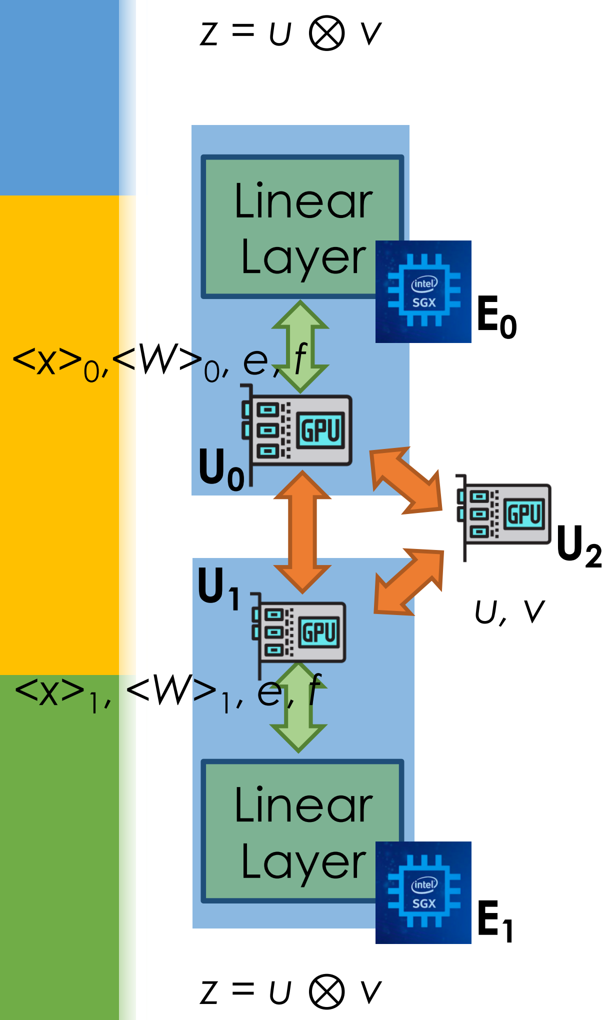
$z = u \otimes v$

$<x>_1, <W>_1, e, f$

- Goal: Compute $y = W \otimes x$
- Without leaking any (W, x, y) to $(\mathbf{U_0}, \mathbf{U_1}, \mathbf{U_2})$
- $\mathbf{E}_{i \in \{0,1\}}$'s Masking & Dispatch:
  - $<W>_i \leftarrow Gen_i(r_W, W)$
  - $<x>_i \leftarrow Gen_i(r_x, x)$
  - $e = W - Rand(r_u) = W - u$
  - $f = x - Rand(r_v) = x - v$

- $Gen_0(r_x, x)$ and $Gen_1(r_x, x)$ are generators for an additive SS of x
- $Rand(r)$ is a secure pseudo-random generator
- $\{r_U, r_V, r_X, r_W\}$ are pre-agreed random seeds (of the generators)

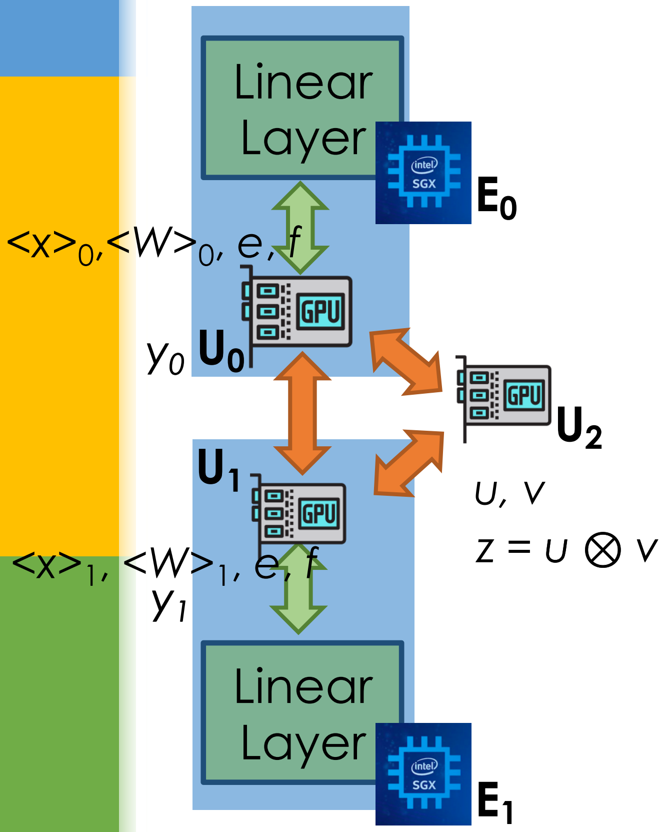
How Goten's GPU-outsourcing works?



- Goal: Compute $y = W \otimes x$
- Without leaking any (W, x, y) to $(\mathbf{U_0}, \mathbf{U_1}, \mathbf{U_2})$
- $\mathbf{E}_{i \in \{0,1\}}$'s Masking & Dispatch:
  - $<W>_i \leftarrow \mathrm{Gen}_i(r_W, W)$
  - $<x>_i \leftarrow \mathrm{Gen}_i(r_x, x)$
  - $e = W - \mathrm{Rand}(r_u) = W - u$
  - $f = x - \mathrm{Rand}(r_v) = x - v$
  - Send $\{<W>_i, <x>_i, e, f\}$ to $\mathbf{U}_i$

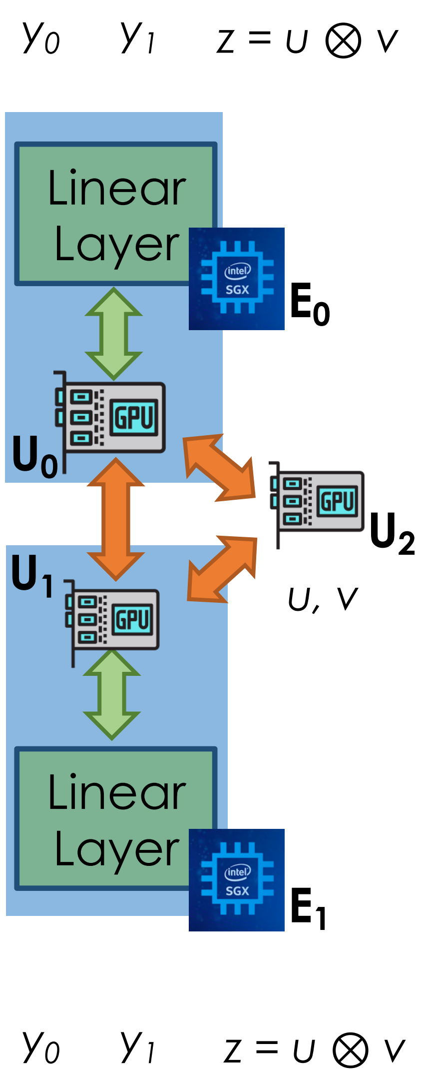
- $\mathrm{Gen}_0(r_x, x)$ and $\mathrm{Gen}_1(r_x, x)$ are generators for an additive SS of x
- $\mathrm{Rand}(r)$ is a secure pseudo-random generator
- $\{r_u, r_v, r_x, r_W\}$ are pre-agreed random seeds (of the generators)

How Goten's GPU-outsourcing works?



- Goal: Compute $y = W \otimes x$
- Without leaking any (W, x, y) to $(\mathbf{U_0}, \mathbf{U_1}, \mathbf{U_2})$
- Outsourced Computation:
  - $\mathbf{U}_0$ computes $y_0 = <W>_0 \otimes f + e \otimes <x>_0$
  - $\mathbf{U}_1$ computes $y_1 = <W>_1 \otimes f + e \otimes <x>_1 - e \otimes f$

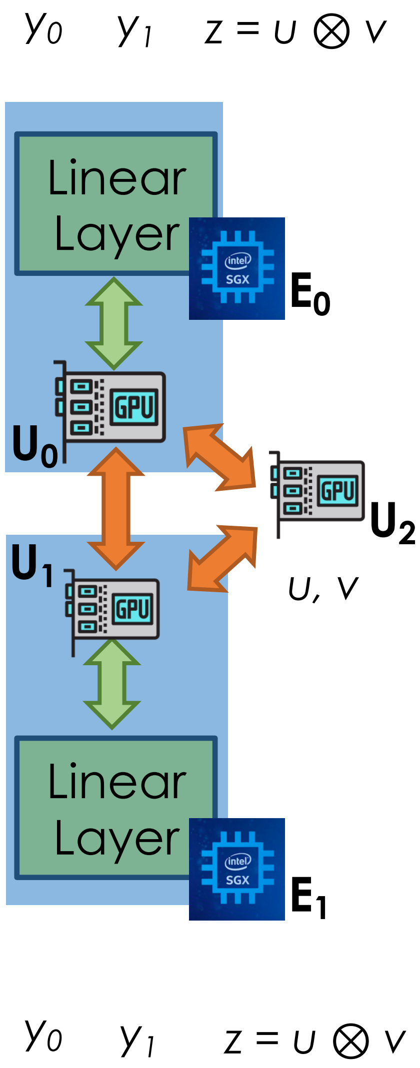
How Goten's GPU-outsourcing works?



- Goal: Compute $y = W \otimes x$
- Without leaking any (W, x, y) to ($\mathbf{U_0}$, $\mathbf{U_1}$, $\mathbf{U_2}$)
- Outsourced Computation:
  - $\mathbf{U}_0$ computes $y_0 = \langle W\rangle_0 \otimes f + e \otimes \langle x\rangle_0$
  - $\mathbf{U}_1$ computes $y_1 = \langle W\rangle_1 \otimes f + e \otimes \langle x\rangle_1 - e \otimes f$
  - $\mathbf{U}_0$ sends y_0 to $\mathbf{E}_0$ and $\mathbf{E}_1$
  - $\mathbf{U}_1$ sends y_1 to $\mathbf{E}_0$ and $\mathbf{E}_1$

- $y_0 = \langle W\rangle_0 \otimes f + e \otimes \langle x\rangle$
- $y_1 = \langle W\rangle_1 \otimes f + e \otimes \langle x\rangle_1 - e \otimes f$

How Goten's GPU-outsourcing works?



- Goal: Compute $y = W \otimes x$
- Without leaking any (W, x, y) to $(\mathbf{U_0}, \mathbf{U_1}, \mathbf{U_2})$
- $\mathbf{E}_{i \in \{0, 1\}}$'s reconstruction of the results:
  - $\mathbf{E}_0$ and $\mathbf{E}_1$ compute $y = y_0 + y_1 + z$

- $y_0 = \langle W \rangle_0 \otimes f + e \otimes \langle x \rangle$
- $y_1 = \langle W \rangle_1 \otimes f + e \otimes \langle x \rangle_1 - e \otimes f$

Correctness of the GPU Outsourcing

- $\mathbf{E_0}$ and $\mathbf{E_1}$ get y
- $y = y_0 + y_1 + z$

  $= \langle W\rangle_0 \otimes f + e \otimes \langle x\rangle_0 + \langle W\rangle_1 \otimes f + e \otimes \langle x\rangle_1 - e \otimes f + u \otimes v$

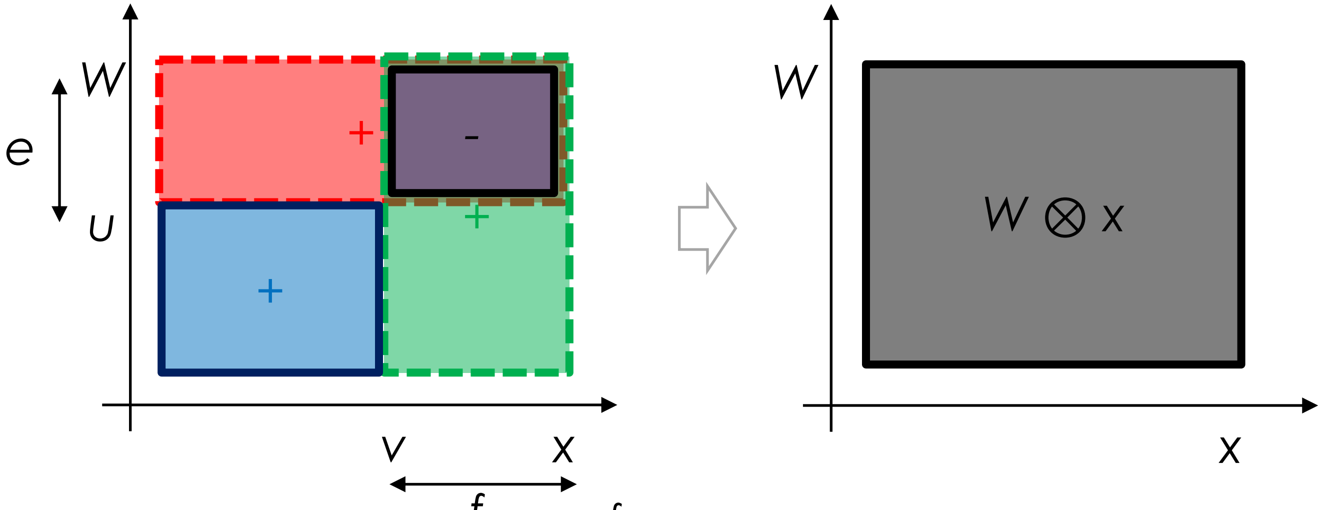
  $= W \otimes f + e \otimes x - e \otimes f + u \otimes v$

  $= W \otimes x$

• $y_0 = \langle W\rangle_0 \otimes f + e \otimes \langle x\rangle$

• $y_1 = \langle W\rangle_1 \otimes f + e \otimes \langle x\rangle_1 - e \otimes f$

- $e = W - u$



- $f = x - v$

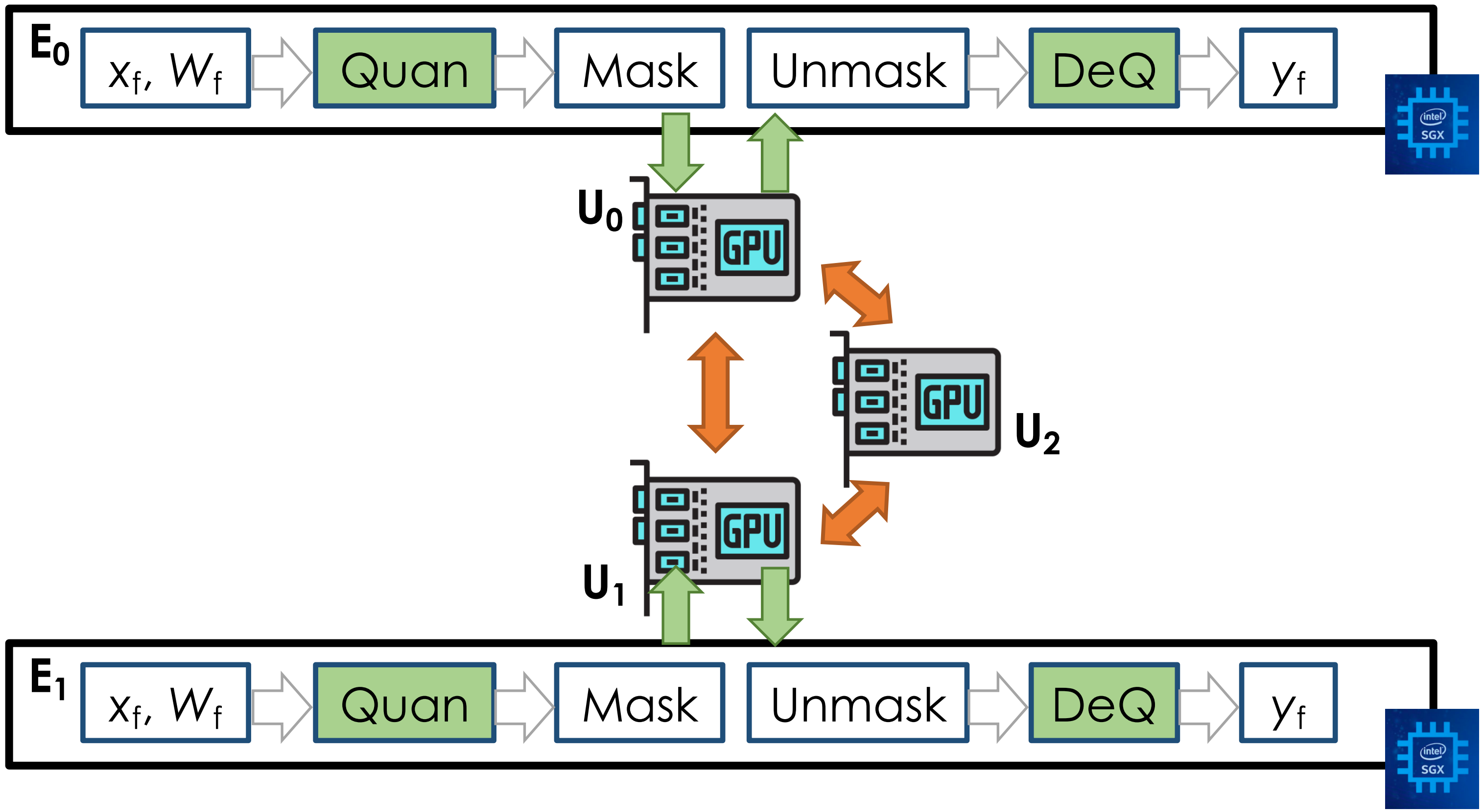
Security of the GPU Outsourcing

- What each non-colluding server sees:
  - $\mathbf{U_0}$: $\langle W \rangle_0$, $\langle x \rangle_0$, e, *f*
  - $\mathbf{U_1}$: $\langle W \rangle_1$, $\langle x \rangle_1$, e, *f*
  - $\mathbf{U_2}$: *u*, *v*
- They are all secret shares or random tensors
  - $\langle W \rangle_{0/1}$ and $\langle x \rangle_{0/1}$ 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 $\mathbf{Z}_q$ (fixed-point integers)
  - Only then $\langle W \rangle_{0/1}$, $\langle x \rangle_{0/1}$, e, *f* are uniformly distributed (over their space)

Quantization for Secure Outsourcing

- (Linear layers') Outsourcing protocol runs over $\mathbf{Z}_q$
  - $y_Q = W_Q \otimes_Q x_Q \bmod q$ (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**!
  - $W_Q = \mathrm{Quan}(W_f; \theta_W)$, $x_Q = \mathrm{Quan}(x_f; \theta_x)$
  - $y_f = \mathrm{DeQ}(y_Q; \theta_W, \theta_x)$

Quantization for Secure Outsourcing



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: $y_f = \text{round}(y_Q \cdot 2^{-8})$
  - θ_W and θ_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
    - $w_Q \approx \text{round}(W_f \cdot 2^8) > q \rightarrow$ 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's *Dynamic* Quantization

- The quantization param. (θ_W, θ_x) varies with the inputs
- Adapt from SWALP[Yang *et al.*]
  - a training scheme for low-bitwidth environment
- θ_w is the magnitude of the maximum values of W
  - $\theta_w = \lfloor \log 2 \circ \max \circ \mathrm{abs}(W) \rfloor$
  - θ_x is also found similarly
- $W_Q = \mathrm{Quan}(W_f; \theta_w) = \mathrm{clip}(\lfloor W \cdot 2^{-\theta_w+6} \rfloor, -2^7, 2^7)$
  - 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: $dx = dy \times W$

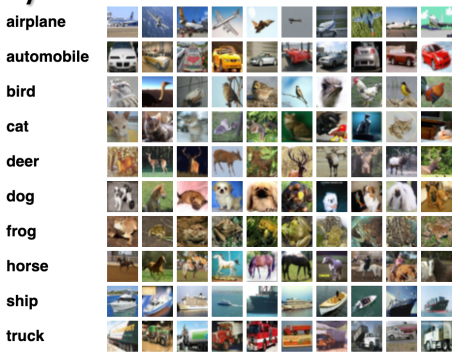
Performance: Training over CIFAR-10

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

Framework	GPU | TEE	NN Architecture	Throughput	Speedup
Falcon	✗ | ✗	VGG-16	1482	132×
CaffeScone[*]	✗ | ✓	VGG-11	28800	6.84×
Goten	✓ | ✓	VGG-11	196733	-

(Images/hour)

- 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.*]

Accuracy	**81%**	**82%**	**83%**	**84%**	**85%**	**86%**
Speedup	8.53×	13.7×	4.27×	6.33×	3.42×	7.28×
Time (min)	1.25	1.56	13.1	16.9	31.2	46.8

Cruz-Roa *et al.*
Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks.
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
  - "**G**PU-**F**riendly **O**blivious and **R**apid **C**lassification **E**ngine"
    - Conditionally Accepted by Usenix Security 2021
- Contact: **`{luciengkl, sherman}@ie.cuhk.edu.hk`**

Goten

- “Understand” (悟) the “sky” (天)

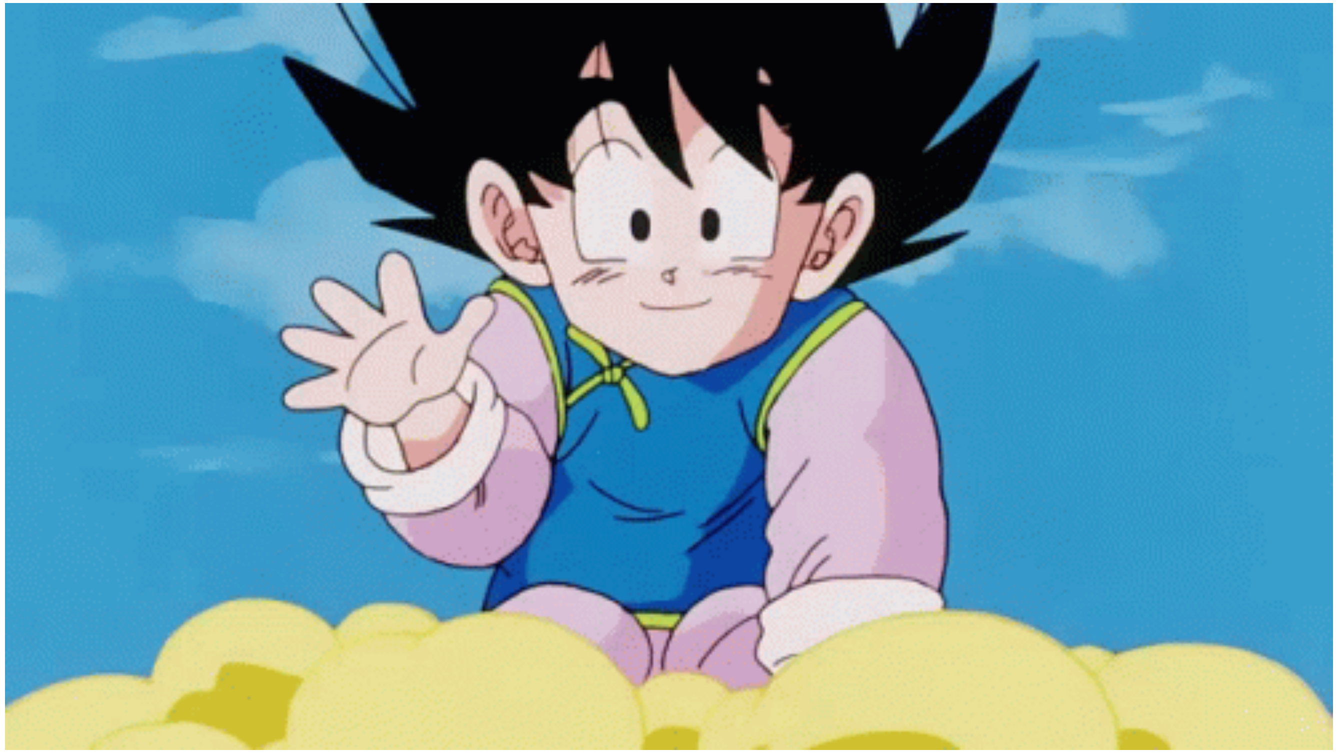


Image Credit: Dragon Ball