CryptGPU

Amithabh A
112101004

June 20, 2024

Introduction

The importance of GPU acceleration

- CNN training could be greatly accelerated through the use of GPUs
- PyTorch and TensorFlow support and rely heavily on GPUs and ASICs (Application-specific integrated circuits) such as Google's tensor processing unit.

Author's Contributions

- CryptGPU, a cryptographic MPC framework built on top of PyTorch and CrypTen. All cryptographic operations (both linear and non-linear) are implemented on the GPU
- Operates on standard 3-party setting, with assumption that all inputs are secret-shared across three servers who execute the MPC protocol.
- Can perform Private inference over ResNet-152 and ImageNet faster than CryptFlow

Introduction

- 36× speed-up for private training of AlexNet on the Tiny ImageNet database compared to Falcon. Whereas it would have taken over a year to privately train Falcon on Tiny ImageNet, CryptGPU is able to do so in a week.
- Usage of Cryptographic protocols that are GPU-friendly and uses GPU parallelism

Cryptography on the GPU

- Leveraging existing CUDA kernels : need a way to embed integer-valued cryptographic operations into (64-bit) floating-point arithmetic that can be operated on by these kernels.

GPU-friendly cryptography

- GPU architecture is optimized simple computations(vector addition and multiplication are fast but operations having conditional statements are slow).
- Cryptographic protocols that solve this are discussed in threat model

Introduction

Systematic evaluation of GPU-based MPC

- Linear layers account for 86% to 99% of the total computational costs of private training in Falcon
- GPU-based approach evaluates the same linear layers with a 25× to 72× speed-up
- Evaluating convolutions on secret-shared data and kernels on GPU is 150× faster than the corr. protocol on the CPU

An ML-friendly approach

- CryptGPU is built on top of CrypTen, which is itself built on top of PyTorch.
- New cryptographic back end that supports computations on secret-shared values while retaining similar front end as PyTorch

System Overview

- Data and model are (arbitrarily) partitioned across three parties
- Clients first secret share their inputs to three independent cloud-service providers. They run the cryptographic protocol on secret-shared inputs.
- CryptGPU's protocols provide security against a single semi-honest corruption

Background

CryptGPU adapt the basic architecture of CrypTen, and make modifications to support three-party protocols based on replicated secret sharing

System Overview

CrypTEN

- CrypTEN provide a secure computing back end for PyTorch while still preserving the PyTorch front end APIs
- MPCTensor in CrpyTEN, functions like a standard PyTorch tensor, except values are secret shared across multiple machines. Internally, CRYPTEN uses n-out-of-n additive secret sharing.
- For Bilinear operations(Matrix Mulitplication and convolutions) CrypTEN uses arithmetic secret sharing over a large ring (e.g., $\mathbb{Z}_{2^{64}}$)

System Overview

Floating point computations

- cryptographic core of CRYPTGPU relies on (additive) replicated secret sharing over the 64-bit ring $\mathbb{Z}_{2^{64}}$.
- The goal is to take advantage of the GPU to accelerate each party's local computation on their individual shares.
- Embed the ring operations over $\mathbb{Z}_{2^{64}}$ into 64-bit floating point operations.

Integer operations using floating-point arithmetic

Exact computation for small values

- 64-bit floating pt. vals have 52 bits of precision - can exactly represent all integers in the interval $[-2^{52}, 2^{52}]$.
- $\forall a, b \in \mathbb{Z}_{2^{64}}[-2^{26}, 2^{26}]$, we can compute the product ab using their floating-point representations

Bilinearity

- Matrix Multiplication and convolution are bilinear. i.e, $(A1 + A2) \circ (B1 + B2) = A1 \circ B1 + A2 \circ B1 + A2 \circ B1 + A2 \circ B2$

System Overview

Evaluation of Bilinear operation

- CryptGPU decomposes each inputs **A**,**B** $\in \mathbb{Z}_{2^{64}}^{n \times m}$ into smaller inputs A_1, ..., A_k and B_1, ..., B_k where A $= \Sigma_{i=1}^{k} 2^{(i-1)w} A_i$ and computes k^2 products $A_I \circ B_j$
- As long as the entries of $A_i \circ$B$_j$ do not exceed 2^{52} in magnitude, all of the above pairwise products are computed
- CryptGPU decomposes each input into k $= 4$ blocks when performing computations using floating-point kernels, where values in each block are represented by w=16-bit value.

Threat Model and Cryptographic Design

Semi Honest Security

.

Let $f : (\{0,1\}^n)^3 \rightarrow (\{0,1\}^m)^3$ be a randomized functionality.

.

Let π be a protocol.

.

π securely computes f in presence of single semi-honest corruption if there exists an efficient simulator S such that for every corrupted party $i \in \{1,2,3\}$ and every input $\mathbf{x} \in (\{0,1\}^n)^3$,

.

$\{output^{\pi}(\mathbf{x}), view_i^{\pi}(\mathbf{x})\} \approx^c \{f(\mathbf{x}), S(i, x_i, f_i(\mathbf{x}))\}$,

.

- $view_i^{\pi}(\mathbf{x})$ is the view of party i in execution π on input x
- $output^{\pi}(\mathbf{x})$ is output of all parties in an execution of π on input x
- $f_i(\mathbf{x})$ denote ith output of x

Threat Model and Cryptographic Design

Computing on secret-shared values

- Two main settings : private inference and private training
- Two secret sharing : standard 3-out-of-3 additive secret sharing and 2-out-of-3 replicated secret sharing
- Modelled both types of secret sharing as pair of algorithms (Share, Reconstruct)

Threat Model and Cryptographic Design

Properties of algorithmic pair (Share, Reconstruct)

- Share$(x) = (x_1, x_2, x_3)$
- Reconstruct(S) takes set of shares. If successful, it outputs $x \in \{0,1\}^n$, and returns $\perp$ otherwise.

Threat Model and Cryptographic Design

Correctness of computations

Threat Model and Cryptographic Design

Private Inference

- $Eval(M, x)$:Problem of evaluating a trained model M on an input x
- Ideal functionality f maps secret shares of an input x and a model M to a secret share of the output $Eval(M, x)$.
- I/p : $((M_1, x_1), (M_2, x_2), (M_3, x_3))$
- Ideal functionality outputs $Share(Eval(M, x))$ where
  - $M \leftarrow \text{Reconstruct}(M_1, M_2, M_3)$
  - $x \leftarrow \text{Reconstruct}(x_1, x_2, x_3)$

Threat Model and Cryptographic Design

Private Training

- Goal is to run a training algorithm Train on some dataset D
- Ideal functionality f maps secret shares of the dataset $(D1, D2, D3)$ to a secret share of the model $Share(Train(D))$ where
  - D $\leftarrow$ Reconstruct$(D1, D2, D3)$.

No party individually learn nothing about input dataset D or resulting learned model Train(D).

Threat Model and Cryptographic Design

B. Cryptographic Building Blocks for Private Inference

- Inference Algo decomposition : linear + pooling + convolution layer and Activation function evaluation(ReLU)

.

GPU-friendly cryptography Protocols that are particularly amenable to GPU acceleration, like protocols that

- Involve conditionals (garbled circuits)
- Require extensive finite field arithmetic

Threat Model and Cryptographic Design

Secret sharing

- We work over the ring $\mathbb{Z}_n$ where $n = 2^k$, $k = 64$
- To secret share $x \in \mathbb{Z}_n$, sample shares $x_1, x_2, x_3 \xleftarrow{R} \mathbb{Z}_n$ such that $x_1 + x_2 + x_3 = x$.
- Default sharing : 2-out-of-3 replicated secret sharing where each party hold a pair of shares: P_i has (x_i, x_{i+1}). It is denoted by $[[x]]^n = (x_1, x_2, x_3)$
- 3-out-of-3 additive secret sharing scheme is used sometimes where party P_i holds x_i

Threat Model and Cryptographic Design

Fixed point representation

- Cryptographic protocols are restricted to computations over discrete domains
- To do:
  - Use a fixed-point encoding of all values
  - Embed the integer valued fixed-point operations in the ring $\mathbb{Z}_n$
- Fixed point encoding of a real value $x \in \mathbb{R}$ is represented by the integer $\lfloor x \cdot 2^t \rceil$ (i.e., the nearest integer to $x \cdot 2^t$) where t is no of bits of precision
- Ring modulus n is chosen to ensure no overflow on integer-valued fixed-point operations (CryptGPU sets $n = 64$)

Threat Model and Cryptographic Design

Protocol Initialization(I didn't understand this quite lot

- Assumption : parties have many independent secret shares of 0.
- F be a pseudorandom function (PRF)
- Each party P_i samples a key k_i and sends it to party P_{i+1}.
- jth secret share of 0 is the triple (z_1, z_2, z_3) where $z_i = F(k_i, j) - F(k_{i-1}, j)$

Linear Operations

- Linear operations on secret-shared data only require local computation
- α, β, γ are public constants
- $[[x]]^n$ and $[[y]]^n$ are secret shared values.
- $[[\alpha x + \beta y + \gamma]]^n = (\alpha x_1 + \beta y_1 + \gamma, \alpha x_2 + \beta y_2, \alpha x_3 + \beta y_3)$
- Each of the parties can compute their respective shares of $[[\alpha x + \beta y + \gamma]]^n$ from their shares of $[[x]]^n$ and $[[y]]^n$ and the public constants

Threat Model and Cryptographic Design

Multiplication

- Multiplication of two secret shared values $[[x]]^n = (x_1, x_2, x_3)$ and $[[y]]^n = (y_1, y_2, y_3)$
  - This yields a 3-out-of-3 additive sharing of z
- Each party locally computes $z_i = x_i y_i + x_{i+1} y_i + x_i y_{i+1}$
- By construction, $z_1 + z_2 + z_3 = xy \in \mathbb{Z}_n$. This yield 3-out-of-3 additive sharing of z
- To obtain replicated shares of z, party P_i sends P_{i+1} a blinded share $z_i + \alpha_i$, where $(\alpha_1, \alpha_2, \alpha_3)$ is a fresh secret sharing of 0(I got it).
- Parties need to rescale z (divide by scaling factor 2^t) after computing the product.
  - Using Π_{trunc1} from ABY^3
  - add something if wanted

Threat Model and Cryptographic Design

Convolutions and Matrix Multiplication(Needs info of Truncation protocol)

- Linear function on secret shared tensors only require local computations.
- Bilinear operation on secret-shared tensors like matrix-multiplication and convolutions are implemented by computing three seperate protocols
- Truncation protocol is applied to result of the product and not after each individual multiplication.
- Impacts on performance :
  - Existing CUDA kernels can be used for matrix products and convolutions
  - Total communication in protocol is proportional to size of the output rather than number of intermediate element-wise multiplications

Most significant bit

- Given an arithmetic secret sharing $[[x]]^n = (x_1, x_2, x_3)$ of x, parties reinterpret it as three binary secret shares $[[x_1]]^2 = (x_1, 0, 0)$, $[[x_2]]^2 = (0, x_2, 0)$, $[[x_3]]^2 = (0, 0, x_3)$ to compute binary shares of the sum $[[x]]^2$, which in particular, yields a binary share of $[[msb(x)]]^2$.
- To recover arithmetic shares of $[[ReLU(x)]]^n$ from $[[x]]^n$, we use bit injection protocol from ABY^3
- Majority of this computation is the evaluation of the addition circuit over binary shares on the GPU
- Evaluating a Boolean addition circuit on secret-shared binary values decomposes into sequence of bitwise AND and XOR operations, which can be computed using GPU kernels

Threat Model and Cryptographic Design

ReLU Activation Function

- $ReLU(x) := max(x, 0)$
- Computation of ReLU function corresponds to computating $msb(x)$ of x

C. Additional Building Blocks for Private Training

- Standard backward Propagation setting with a softmax/cross entropy loss function optimized using Stochastic Gradient Descent(SGD)

Summary of Work :

- Classification tasks with d target classes
- Each iteration of SGD takes
  - input $x \in \mathbb{R}^m$
  - One-hot encoding of target vector $y \in \{0,1\}^d$, $y_i = 1$ if x belong to class i and $y_i = 0$ otherwise.
- Cross entropy computation :

$$l_{CE}(\mathbf{x};\mathbf{y}) := -\Sigma_{i\in[d]}\ y_i\ log\ \widetilde{z}_i \tag{1}$$

where $\widetilde{z} \leftarrow \text{softmax}(\mathbf{z})$, $\mathbf{z} \leftarrow \text{Eval}(M, x)$ and M is the current model

Threat Model and Cryptographic Design

- For a vector $x \in \mathbb{R}^d$, softmax function $\text{softmax}(\mathbf{x}_i) := e^{x_i} / \sum_{i \in [d]} e^{x_i}$
- Gradient of l_{CE} for output layer z is

$$\nabla_{\mathbf{z}} l_{CE} = softmax(\mathbf{z}) - \mathbf{y} \tag{2}$$

- To compute $[[z]]^n$ from $[[x]]^n$ and $[[M]]^n$, use private inference protocol.

Threat Model and Cryptographic Design

Softmax

- Evaluate the softmax on the normalized vector ($\mathbf{x}$ - $\max_i x_i$) to avoid numeric imprecision from evaluating the exponential function in the softmax function
- softmax($\mathbf{x}$ - $\max_i x_i$) = softmax($\mathbf{x}$)
- advantage : all inputs to softmax function are at most 0, and the denominator is contained in the interval [1, d]
- Protocol for computing softmax on secret-shared values needs following protocols :
  - Exponential function evaluation
  - Division
  - Maximum over a vector of secret shared values

Threat Model and Cryptographic Design

1. Exponentiation Approximating e^x needed to compute softmax with its limit characterisation f_m:

$$f_m(x) := (1 + \frac{x}{m})^m \tag{3}$$

Using a Taylor expansion for the function $\ln(1 + x)$ and assuming that $|x| < m$,

$$\frac{f_m(x)}{e^x} = \frac{e^{m \ln(1+x/m)}}{e^x} = e^{-O(x^2/m)} \tag{4}$$

Compared to Taylor approximation, limit-based approximation f_m is more efficient to evaluate (in terms of the number of multiplications) and more robust for handling large negative inputs that may arise in the computation.

Division

- To compute $[[x/y]]^n$, compute the reciprocal $[[1/y]]^n$ and compute the quotient, $[[x]]^n$ and $[[y]]^n$ are secret shared values, $1 \leq y \leq Y$ for some bound Y
- Neuton-Raphson Algorithm for approximating value of $1/y$
  Initial guess : z_0
  Iterative step :

$$z_i \leftarrow 2z_{i-1} - yz_{i-1}^2 \tag{5}$$

Threat Model and Cryptographic Design

- In CryptGPU : $z_0 = 1/Y$
  - Provides highly accurate estimation for $1/y$ for all $y \in [1, Y]$ using $O(\log Y)$ iterations of Neuton's algorithm

$$error_i = |1/y - z_i| = \frac{1}{y}|1 - z_i y| \leq \epsilon_i \tag{6}$$

  where $\epsilon_i = |1 - z_i y|$
- Maximum error after i iteration : $(1 - 1/Y)^{2^i} \leq e^{-2^i/Y}$
- add rest if you want to

Threat Model and Cryptographic Design

Maximum

- max value $[[max_i x_i]]^n$ from $[[x]]^n$ where $x \in \mathbb{R}$
- Using tree of comparisons to reduce round complexity to log m where pairs of elements are compared and larger val in each pair advance to next round
- Comparing fixed point vals $[[x]]^n$, $[[y]]^n$ is equivalent to computing msb of difference ($[[msb(x - y)]]^n$)

Derivative of ReLU

- 0 if $x < 0$
- 1 if $x > 0$

System Implementation and Evaluation

Point-to-point communication

- default communication mode in PyTorch : **Broadcast** mode
- Broadcast channel between each pair of parties functions as a **point-to-point channel** between the parties.

Pseudorandom generators on the GPU

Paper uses **AES** as PRF in their protocol, by using `torchcsprng` PyTorch C++/CUDA extension.

System Implementation and Evaluation

Deep Learning Datasets

- **MNIST**
  - dataset for handwritten digit recognition.
  - benchmark in many privacy-preserving ML systems
- **CIFAR 10**
  - Dataset with 60,000 32x32 RGB images split evenly across 10 classes.
- **Tiny ImageNet**
  - Modified subset of the ImageNet dataset
  - 100,000 64 x 64 RGB training images, 10,000 testing images, split across 200 classes.
  - More challenging than CIFAR 10
- **ImageNet**
  - Large Scale Visual recognition Dataset
  - Standard benchmark for evaluating the classification performance of computer vision models
  - 1000 classes,
  - CrypTFlow is the only prior system for privacy-preserving machine learning that demonstrates performance at scale of ImageNet.

System Implementation and Evaluation

Deep Learning Models

- **LeNet**
  - Handwritten digit recognition
  - Shallow network with 2 convolutional layers, 2 average pooling layers, and 2 fully connected layers
  - Uses the hyperbolic tangent (tanh) as its activation function.
- **AlexNet**
  - 5 convolutional layers, 3 max pooling layers, and 2 fully connected layers for a total of 61 million parameters
  - Uses ReLU as its activation function.
- **VCG-16**
  - Uses 16 layers consisting of convolution, ReLU, max pooling, and fully-connected layers
- **ResNet**
  - Introduces skip-connections that addresses the vanishing gradient problem when training deep neural network models.
  - Enjoyed wide adoption in the computer vision community.

System Implementation and Evaluation

Architectural Adjustments AlexNet and VGG-16 on small datasets
AlexNet and VGG-16 are not directly compatible with smaller inputs since they were designed for ImageNet. So the architecture is modified.

- **For AlexNet**
  - Drop the final max pooling layer for CIFAR-10
  - Adjust no. of neurons to 256-256-10 for CIFAR-10
  - Adjust no. of neurons to 1024- 1024-200 for Tiny ImageNet.
- **For VCG-16**
  - Adjust no. of neurons to 256-256-10 for CIFAR-10
  - Adjust no. of neurons to 512- 512-200 for Tiny ImageNet
- **Average pooling**
  - Uses 16 layers consisting of convolution, ReLU, max pooling, and fully-connected layers

Activation Functions

- LeNet uses the hyperbolic tangent function tanh
- CryptGPU does not support evaluating the tanh function and modern networks primarily use ReLU as their activation function.
- They replaced tanh with ReLU in their experiments with LeNet.

Average Pooling

- Pooling : standard way to down-sample outputs of conv. layers in CNN.
- Pooling layer accumulates o/p of conv. layers by replacing feature map window with average of values(avg pooling) / max of values(max pooling)
- Avg Pooling is a linear operation whereas max pooling is a highly non-linear operation
- will talk more later.

Protocol instantiation

1. Fixed-point precision
   - Secret sharing scheme over 64-bit ring $\mathbb{Z}_{64}$
   - $t = 20$ bits of fractional precision
2. Exponentiation
   - f_m, $m = 2^9 = 512$
   - Evaluating f_m requires $\log m = 9$ rounds of multiplication
3. Division
   - Private division protocol to compute $[[1/y]]^n$ where $y \in [1, Y]$
   - $Y \leq 200$ for all our datasets
     - 13 iterations of Newton-Raphson

System Implementation and Evaluation

Comparisons with prior work

- Comparing performance of CryptGPU against Falcon and CryptFlow (only privacy preserving machine learning frameworks that can handle neural networks at the scale of AlexNet on large datasets)

Private Inference

	LeNet (MNIST)		AlexNet (CIFAR)		VGG-16 (CIFAR)		AlexNet (TI)		VGG-16 (TI)	
	Time	Comm. (MB)	Time	Comm. (MB)	Time	Comm. (MB)	Time	Comm. (MB)	Time	Comm. (MB)
FALCON	0.038	2.29	0.11	4.02	1.44	40.45	0.34	16.23	8.61	161.71
CRYPTGPU	0.38	3.00	0.91	2.43	2.14	56.2	0.95	13.97	2.30	224.5
Plaintext	0.0007	—	0.0012	—	0.0024	—	0.0012	—	0.0024	—
	AlexNet (ImageNet)		VGG (ImageNet)		ResNet-50 (ImageNet)		ResNet-101 (ImageNet)		ResNet-152 (ImageNet)	
	Time	Comm. (GB)	Time	Comm. (GB)	Time	Comm. (GB)	Time	Comm. (GB)	Time	Comm. (GB)
CRYPTFLOW	—	—	—	—	25.9	6.9	40*	10.5*	60*	14.5*
CRYPTGPU	1.52	0.24	9.44	2.75	9.31	3.08	17.62	4.64	25.77	6.56
Plaintext	0.0013	—	0.0024	—	0.011	—	0.021	—	0.031	—

System Implementation and Evaluation

- Falcon outperforms CryptGPU for shallow networks and small datasets, but for larger dataset and deep models, CryptGPU is faster
- CryptGPU is 2.2x faster than CrypTFlow on doing private inference on ResNet-152 network
- 1000x gap in performance compared to plaintext inference on the GPU

Batch Private Inference

	$k = 1$		$k = 64$	
	Time	Comm.	Time	Comm.
AlexNet	0.91	0.002	1.09	0.16
VGG-16	2.14	0.056	11.76	3.60

	$k = 1$		$k = 8$	
	Time	Comm.	Time	Comm.
ResNet-50	9.31	3.08	42.99	24.7
ResNet-101	17.62	4.64	72.99	37.2
ResNet-152	25.77	6.56	105.20	52.5

- Leverage GPU parallelism by processing batch of images so that cost of inference could be amortized.

System Implementation and Evaluation

Private Training

- GPUs to have a larger advantage in the setting of private training
- CryptGPU achieves a considerable speedup over Falcon, especially over larger models and datasets
- A Single iteration of (private) backpropagation completes in 11.30s with CRYPTGPU and 6.9 minutes using FALCON to train AlexNet on Tiny ImageNet set.
- Privately training AlexNet on Tiny ImageNet takes 10 days using CryptGPU while it would takes 375 days using Falcon (assuming 100 epochs over the training set).
- There still remains a large gap (roughly 2000x) between the costs of private training and plaintext training (on the GPU)

Private Training breakdown : Comparison between Falcon(CPU based Protocol) and CryptGPU(GPU based Protcol)

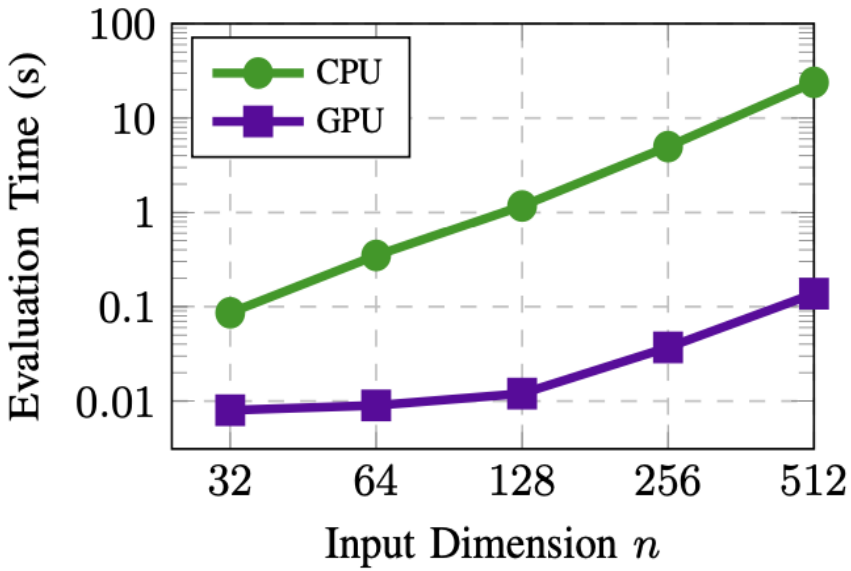
- Computation of the linear layers in CryptGPU between 25x and 70x faster than Falcon
- For Pooling layers, Falcon uses Max Pooling and CryptGPU uses Avg Pooling. CryptGPU maintains a (significant) performance edge even if we exclude the cost of the pooling layers from the running time of FALCON.
- For ReLU layers, CPU-based protocol in Falcon compares very favorably with ReLU protocol in CRYPTGPU. The Protocol performance will improve by having ReLU protocol that takes advantage of GPU parallelism.

Comparing Falcon and CryptGPU

	Linear		Pooling		ReLU		Softmax	
	FALCON	CRYPTGPU	FALCON	CRYPTGPU	FALCON	CRYPTGPU	FALCON	CRYPTGPU
LeNet (MNIST)	13.07	0.49	1.34	0.076	0.47	1.00	—	0.53
AlexNet (CIFAR)	59.23	0.86	2.65	0.077	0.41	1.33	—	0.55
VGG-16 (CIFAR)*	355.16	6.33	2.86	0.21	5.40	4.74	—	0.53
AlexNet (TI)	402.45	5.60	10.20	0.37	1.92	4.16	—	1.04
VGG-16 (TI)†	355.84	7.61	2.87	0.32	5.37	4.73	—	0.98

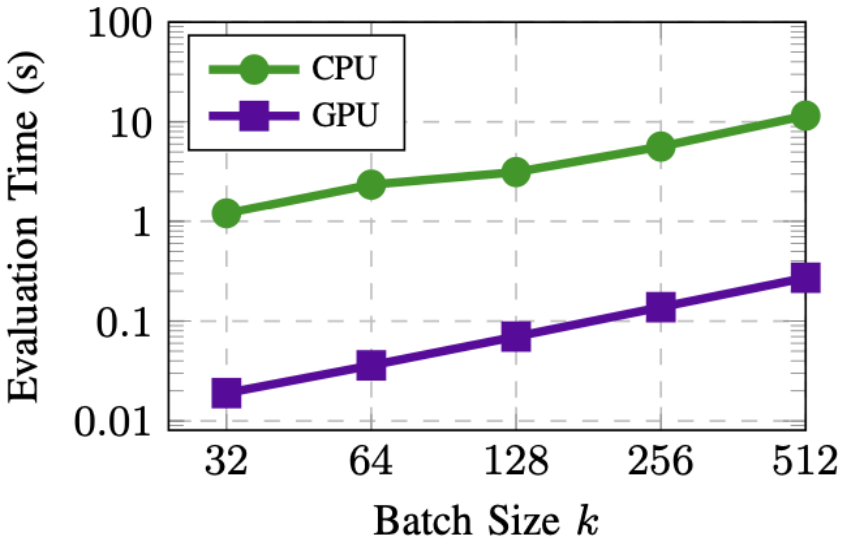
- Runtime (in seconds) of Falcon and CryptGPU for evaluating the linear, pooling, ReLU, and softmax layers for different models and datasets during private training.
- Linear layers include the convolution and the fully-connected layers.
- Pooling layer refers to max pooling in Falcon, and avg. pooling in CryptGPU.
- Implementation of Falcon donot support softmax evaluation (and correspondingly, gradient computation for the output layer).

Comparing Falcon and CryptGPU



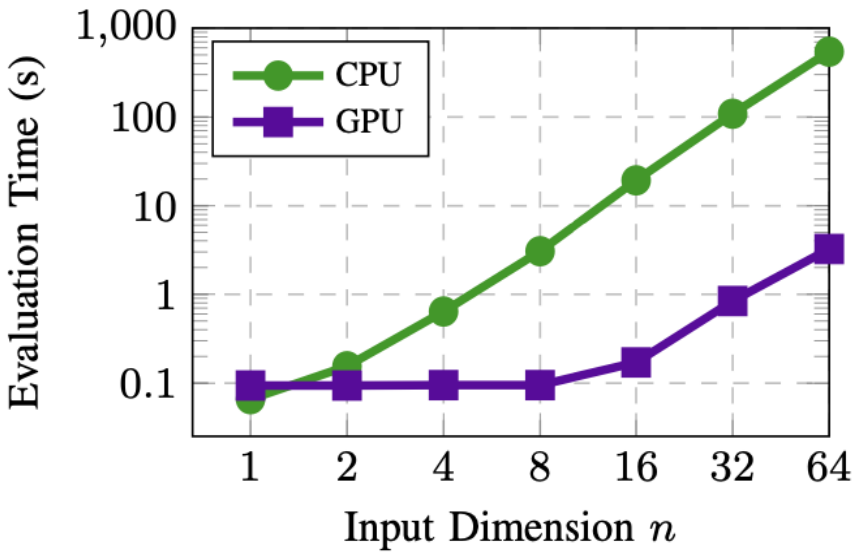
Convolution on an $n \times n \times 3$ input with a 11×11 kernel, 64 output channels, 4×4 stride, and 2×2 padding

Comparing Falcon and CryptGPU



Convolution on batch of k 32 $\times$ 32 $\times$ 3 inputs with an 11 $\times$ 11 kernel, 64 output channels, 4 4 stride, and 2 $\times$ 2 padding.

Comparing Falcon and CryptGPU



Convolution on an n×n×512 input with a 3×3 kernel, 512 output channels, 1×1 stride, and 1×1 padding.

Private ReLU: GPU vs. CPU

- Delphi executed the non-linear steps (e.g., ReLU computations) on the CPU. CryptGPU takes advantage of GPU parallelism to accelerate non-linear computations with chosen set of cryptographic protocols.
- CryptGPU has 16× speedup when evaluating ReLU on a block of 256,000 secret shared inputs
- As inputs scale up to a block with 32 million inputs (250 MB of data), there is a 9× speedup on the GPU

Average Pooling vs Max pooling

Attempt to evaluate whether the choice of pooling makes a significant difference on model performance.

- Trained AlexNet and VGG-16 networks over CIFAR-10 dataset where all max pooling layers is replaced with average pooling layers
- 3% drop in accuracy (from 76% to 73%) for AlexNet and a 1% increase in accuracy with VGG-16 (from 82% to 83%)
- Average pooling in place of Max pooling donot make degradation of model performance significantly.