Telekine: Secure Computing with Cloud GPUs

NSDI 2020

Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J. Rossbach, Emmett Witchel
Trusting the cloud provider is difficult
Trusting the cloud provider is difficult

- Attackers can exploit system bugs to steal data
Trusting the cloud provider is difficult

- Attackers can exploit system bugs to steal data
- The cloud provider has their own interests (e.g., monetizing user data)
Trusting the cloud provider is difficult

- Attackers can exploit system bugs to steal data
- The cloud provider has their own interests (e.g., monetizing user data)
- Many administrators; some may be malicious
Avoiding trust in the cloud provider is difficult.
Avoiding trust in the cloud provider is difficult.

OS/Hypervisor allows provider to see user’s secret data.
Avoiding trust in the cloud provider is difficult.

Legend:
- Trusted
- Untrusted
- Data

OS/Hypervisor allows provider to see user’s secret data.
Introduce TEEs to isolate computation
(TEE is Trusted Execution Environment)

• TEEs cannot be bypassed by software
  - Hardware root of trust (e.g., SGX on Intel, TrustZone on ARM)

• Protect communication from the provider with cryptography

• Research proposals exist for GPU TEEs
  - Graviton [OSDI’18], HIX [ASPLOS’19]
    - Performance critical hardware unchanged
Introduce TEEs to isolate computation
(TEE is Trusted Execution Environment)

• TEEs cannot be bypassed by software
  - Hardware root of trust (e.g., SGX on Intel, TrustZone on ARM)

• Protect communication from the provider with cryptography

• Research proposals exist for GPU TEEs
  - Graviton [OSDI’18], HIX [ASPLOS’19]
  - Performance critical hardware unchanged
TEEs have limitations
TEEs have limitations

Tensorflow/MXNet

Legend:
- Trusted
- Untrusted
- Data

CPU TEEs:
- Cache Side Channels
- Spectre attacks

Recognition Results

Neural Net Computation

TEE

TEE
TEEs have limitations

Mitigations require heroic effort, especially for complex software
Telekine addresses TEE limitations

Legend:
- Trusted
- Untrusted
- Data

Diagram:
- Tensorflow/MXNet
- Recognition Results
- GPU API Proxy
- Neural Net Computation
- TEE
Telekine uses API-remoting instead of CPU TEEs

- Interpose on GPU API calls
- Application does not have to be modified, user does not need GPU
- Turn every API call into an RPC, executed by the remote machine
- Traffic is encrypted/authenticated; the proxy does not need protection
TEEs still have limitations

Legend:
- Trusted
- Untrusted
- Data

<table>
<thead>
<tr>
<th>GPU API Proxy</th>
<th>Neural Net Computation</th>
<th>TEE</th>
</tr>
</thead>
</table>

- Recognition Results

- Tensorflow/MXNet

- Data

- Trusted

- Untrusted
TEEs still have limitations

Legend:
- Trusted
- Untrusted
- Data

All TEEs: Communication timing attacks
Telekine addresses communication timing channels

- TEEs do not consider communication side channels
  - Securing the processor (CPU/GPU) does not secure communication
- GPU programming paradigm features frequent communication
  - CPU-to-CPU communication is also vulnerable
- Communication patterns tend to leak timing information
  - E.g., GPU kernel execution time
In the rest of the talk we will answer:

• Can information be extracted from GPU communication patterns?

• How does Telekine remove that information?

• What are Telekine’s overheads?
In the rest of the talk we will answer:

• Can information be extracted from GPU communication patterns? **Yes, we demonstrate a communication timing attack**

• How does Telekine remove that information?

• What are Telekine’s overheads?
In the rest of the talk we will answer:

- Can information be extracted from GPU communication patterns?  
  **Yes, we demonstrate a communication timing attack**

- How does Telekine remove that information?
  **Replace GPU streams with new data-oblivious streams**

- What are Telekine’s overheads?
In the rest of the talk we will answer:

• Can information be extracted from GPU communication patterns?
  
  **Yes, we demonstrate a communication timing attack**

• How does Telekine remove that information?
  
  **Replace GPU streams with new data-oblivious streams**

• What are Telekine’s overheads?
  
  **Overheads are reasonable: ~20% for neural network training**
In the rest of the talk we will answer:

- Can information be extracted from GPU communication patterns?
  
  Yes, we demonstrate a communication timing attack

- How does Telekine remove that information?
  
  Replace GPU streams with new data-oblivious streams

- What are Telekine’s overheads?
  
  Overheads are reasonable: ~20% for neural network training
Expanded image recognition

Legend:
- Trusted
- Untrusted
- Data
Expanded image recognition

Tensorflow/MXNet

GPU Stream

Legend:
- Trusted
- Untrusted
- Data

GPU TEE

GPU API
Proxy

Neural Net
State/Code
Expanded image recognition

Tensorflow/MXNet

memcpy()

GPU Stream

Legend:
- Trusted
- Untrusted
- Data

GPU TEE

GPU API Proxy

Neural Net State/Code
Expanded image recognition

Tensorflow/MXNet

memcpy()

GPU Stream

Legend:
- Trusted
- Untrusted
- Data

GPU TEE

GPU API Proxy

Neural Net State/Code
Expanded image recognition

Tensorflow/MXNet

memcpy()

launchKernel(1)

launchKernel(n)

GPU Stream

GPU TEE

Neural Net State/Code

Legend:
- Trusted
- Untrusted
- Data
Expanded image recognition

Tensorflow/MXNet

memcpy()

launchKernel(1)

launchKernel(n)

GPU Stream

Legend:

Trusted

Untrusted

Data

GPU TEE

Neural Net

State/Code

GPU API Proxy

Start
Expanded image recognition

- Tensorflow/MXNet
  - memcpy()
  - launchKernel(1)
  - ...
  - launchKernel(n)

- GPU Stream

- GPU TEE
  - Neural Net
  - State/Code

Legend:
- Trusted
- Untrusted
- Data
Expanded image recognition

```
Tensorflow/MXNet

memcpy()

launchKernel(1)

launchKernel(n)

GPU Stream

GPU TEE

Neural Net

State/Code

Legend: Trusted Untrusted Data

GPU API Proxy

Start

Done

Start

Done
```
Expanded image recognition

Tensorflow/MXNet

memcpy()

launchKernel(1)

launchKernel(n)

GPU Stream

GPU TEE

Neural Net

State/Code

Legend:

Trusted

Untrusted

Data

Recognition

Results

GPU API Proxy

Start

Done

Start

Done

Trusted

Untrusted

Data

Start

Done
Expanded image recognition

Tensorflow/MXNet

memcpy()

launchKernel(1)

launchKernel(n)

memcpy()

GPU Stream

GPU TEE

Neural Net State/Code

GPU API Proxy

Legend: Trusted | Untrusted

Data

Recognition Results

[Encrypted Result]
Expanded image recognition

Tensorflow/MXNet

memcpy()

launchKernel(1)

.

launchKernel(n)

memcpy()

GPU Stream

GPU TEE

Neural Net

State/Code

Legend:

Trusted

Untrusted

Data

GPU raises interrupt on kernel completion
Expanded image recognition

Tensorflow/MXNet

memcpy()

launchKernel(1)

:  

launchKernel(n)

memcpy()

GPU Stream

Neural Net State/Code

GPU API Proxy

Recognition Results

Legend:

Trusted

Untrusted

Data

Kernel execution times!

GPU raises interrupt on kernel completion
Information gained from kernel execution time

Application

Start

Kernel timing classifier

Done

Attack

Image class
Information gained from kernel execution time
Information gained from kernel execution time
Information gained from kernel execution time
Information gained from kernel execution time

![Diagram showing the process]

- **Application**
- **Attack**: Start → Kernel timing classifier → Done
- **Image class**: Classification Accuracy

Graph: Classification Accuracy vs. Number of Classes

- Orange line: GPU Kernel timing classifier
- Black line: Random Guess

Accuracy (%)

<table>
<thead>
<tr>
<th>Number of Classes</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>
Information gained from kernel execution time

Classification Accuracy

Number of Classes

Accuracy (%)

- GPU Kernel timing classifier
- Random Guess

1.6X

Start

Done

Application

Image class

Kernel timing classifier

Start

Done

Attack
In the rest of the talk we will answer:

• Can information be extracted from GPU communication patterns? 
  Yes, we demonstrate a communication timing attack

• How does Telekine remove that information? 
  Replace GPU streams with new data-oblivious streams

• What are Telekine’s overheads? 
  Overheads are reasonable: ~20% for neural network training
In the rest of the talk we will answer:

• Can information be extracted from GPU communication patterns? **Yes, we demonstrate a communication timing attack**

• How does Telekine remove that information? **Replace GPU streams with new data-oblivious streams**

• What are Telekine’s overheads? **Overheads are reasonable: ~20% for neural network training**
Timing information is abundant

Tensorflow/MXNet

memcpy()

launchKernel(1)

launchKernel(n)

memcpy()

GPU Stream

GPU TEE

Neural Net

State/Code

GPU API Proxy

Legend:

Trusted

Untrusted

Data

Recognition Results

Recognition Results
Timing information is abundant
Timing information is abundant

Legend:
- Trusted
- Untrusted
- Data

Application

memcpy()

GPU Stream

GPU API Proxy

GPU TEE
Timing information is abundant

Legend:
- Trusted
- Untrusted
- Data

Application

memcpy()

launchKernel()

GPU Stream

GPU API Proxy

GPU TEE
Timing information is abundant

GPU API Proxy

Legend:
- Trusted
- Untrusted
- Data

Application
- memcpy()
- launchKernel()
- memcpy()

GPU Stream

GPU TEE
Timing information is abundant

Legend:
- Trusted
- Untrusted
- Data

GPU raises interrupt on kernel completion
Timing information is abundant

Legend:
- Trusted
- Untrusted
- Data

Operations are distinguishable because of the hardware they use

GPU raises interrupt on kernel completion
Timing information is abundant

Other potential timing channel sources:
- Commands may be different sizes
- Application’s API use pattern may depend on secret data

Operations are distinguishable because of the hardware they use

GPU raises interrupt on kernel completion
Data-oblivious streams

Legend:
- Trusted
- Untrusted
- Data

Application

memcpy()

launchKernel()

memcpy()

GPU Stream

GPU API Proxy

GPU TEE
Data-obliviuous streams

Legend:
- Trusted
- Untrusted
- Data

Application

memcpy()

launchKernel()

memcpy()

GPU Stream

GPU API Proxy

GPU TEE
Data-oblivious streams

Legend:
- Trusted
- Untrusted
- Data

Application
- memcpy()
- launchKernel()
- memcpy()

Data-oblivious Stream

GPU API Proxy

GPU TEE
Data-oblivious streams

Application

memcpy()

launchKernel()

memcpy()

Data-oblivious Stream

LibTelekine

GPU API Proxy

Legend:

<table>
<thead>
<tr>
<th>Trusted</th>
<th>Untrusted</th>
<th>Data</th>
</tr>
</thead>
</table>

GPU TEE
Data-oblivious streams

Legend:
- Trusted
- Untrusted
- Data

Application
- memcpy()
- launchKernel()
- memcpy()

LibTelekine
- memcpy queue
- launchKernel queue

GPU API Proxy

GPU TEE
Data-oblivious streams

Legend:
- Trusted
- Untrusted
- Data

Application
- memcpy()
- launchKernel()

LibTelekine
- memcpy queue
- GPU stream
- launchKernel queue

GPU API Proxy
- DMA

GPU TEE
Data-oblivious streams

Legend:
- Trusted
- Untrusted
- Data

Applications call `memcpy()` and `launchKernel()` functions, which are then handled by `LibTelekine`. The `memcpy` queue and `launchKernel` queue are used to manage data transfers.

Data-oblivious streams are managed through the GPU API Proxy, which interacts with the GPU TEE. Additional components include DMA and MMIO.
Data-oblivious streams

• Divide commands by type so they can be scheduled independently
  - Adversary sees two independent streams of operations
  - Telekine manages data dependencies between types

• Split and pad commands as necessary; enforce a uniform size

• Queue commands and send them (or no-ops) out deterministically
  - E.g., launch 32 kernels every 15ms, memcpy 1MB both directions every 30ms
Data-oblivious streams

- Divide commands by type so they can be scheduled independently
  - Adversary sees two independent streams of operations
  - Telekine manages data dependencies between types

- Split and pad commands as necessary; enforce a uniform size

- Queue commands and send them (or no-ops) out deterministically
  - E.g., launch 32 kernels every 15ms, memcpy 1MB both directions every 30ms
In the rest of the talk we will answer:

• Can information be extracted from GPU communication patterns?
  *Yes, we demonstrate a communication timing attack*

• How does Telekine remove that information?
  *Replace GPU streams with new data-oblivious streams*

• What are Telekine’s overheads?
  *Overheads are reasonable: ~20% for neural network training*
In the rest of the talk we will answer:

• Can information be extracted from GPU communication patterns?
  Yes, we demonstrate a communication timing attack

• How does Telekine remove that information?
  Replace GPU streams with new data-oblivious streams

• What are Telekine’s overheads?
  Overheads are reasonable: ~20% for neural network training
Testbeds

The cloud machine (Austin, Texas):
Intel i9-9900K, 8 cores @3.60GHz
32GB of RAM
Radeon RX VEGA 64 GPU with 8GB of RAM

Geo-distributed client (Dallas, Texas):
Vultr cloud VM, 8 vCPUs
32GB of RAM

Client (Austin, Texas):
Intel Xeon E3-1270 v6, 4 cores @3.8GHz
32GB of RAM

(RTT is “Roundtrip Time”)
Testbeds

The cloud machine (Austin, Texas):
- Intel i9-9900K, 8 cores @3.60GHz
- 32GB of RAM
- Radeon RX VEGA 64 GPU with 8GB of RAM

Client (Austin, Texas):
- Intel Xeon E3-1270 v6, 4 cores @3.8GHz
- 32GB of RAM

Geo-distributed client (Dallas, Texas):
- Vultr cloud VM, 8 vCPUs
- 32GB of RAM

Real WAN testbed

877Mbps, 12ms RTT

1Gbps, various RTTs

(RTT is “Roundtrip Time”)
Testbeds

The cloud machine (Austin, Texas):
- Intel i9-9900K, 8 cores @3.60GHz
- 32GB of RAM
- Radeon RX VEGA 64 GPU with 8GB of RAM

Client (Austin, Texas):
- Intel Xeon E3-1270 v6, 4 cores @3.8GHz
- 32GB of RAM

Geo-distributed client (Dallas, Texas):
- Vultr cloud VM, 8 vCPUS
- 32GB of RAM

(877Mbps, 12ms RTT)

Client (Austin, Texas):
- Intel Xeon E3-1270 v6, 4 cores @3.8GHz
- 32GB of RAM

1Gbps, various RTTs

(RTT is “Roundtrip Time”)

Real WAN testbed

Simulated WAN testbed
We compare Telekine to an insecure baseline: running on the GPU server without protections

Workloads:

• Data movement vs. GPU work microbenchmark

• Neural net inference on MXNet:
  - ResNet50 [He et. al 2016], InceptionV3 [Szegedy et. al 2016], DenseNet [Huang et. al 2017]

• Neural net training on MXNet:
  - (Same networks as above)

• Graph analytics on Galois:
  - BFS, PageRank, SSSP (across 1 and 2 GPUs)
We compare Telekine to an insecure baseline: running on the GPU server without protections

Workloads:

• Data movement vs. GPU work microbenchmark

• Neural net inference on MXNet:
  - ResNet50[He et. al 2016], InceptionV3[Szegedy et. al 2016], DenseNet [Huang et. al 2017]

• Neural net training on MXNet:
  - (Same networks as above)

• Graph analytics on Galois:
  - BFS, PageRank, SSSP (across 1 and 2 GPUs)
MXNet neural net inference (Real WAN)

- User sends a batch of images to be classified
- Baseline: user sends batch to remote MXNet
- Telekine: user sends batch to local MXNet
  - Telekine remotes computation to the GPU

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>DenseNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MXNet neural net inference (Real WAN)

- User sends a batch of images to be classified
- Baseline: user sends batch to remote MXNet
- Telekine: user sends batch to local MXNet
  - Telekine remotes computation to the GPU

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>DenseNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.0X</td>
<td>6.6X</td>
<td>7.7X</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MXNet neural net inference (Real WAN)

- User sends a batch of images to be classified
- Baseline: user sends batch to remote MXNet
- Telekine: user sends batch to local MXNet
  - Telekine remotes computation to the GPU

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>DenseNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.0X</td>
<td>6.6X</td>
<td>7.7X</td>
</tr>
<tr>
<td>8</td>
<td>3.4X</td>
<td>2.2X</td>
<td>2.5X</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MXNet neural net inference (Real WAN)

• User sends a batch of images to be classified
• Baseline: user sends batch to remote MXNet
• Telekine: user sends batch to local MXNet
  - Telekine remotes computation to the GPU

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>DenseNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.0X</td>
<td>6.6X</td>
<td>7.7X</td>
</tr>
<tr>
<td>8</td>
<td>3.4X</td>
<td>2.2X</td>
<td>2.5X</td>
</tr>
<tr>
<td>64</td>
<td>1.0X</td>
<td>1.1X</td>
<td>1.0X</td>
</tr>
</tbody>
</table>
MXNet neural net training (Real WAN)

- Large dataset of images, processed in batches of size 64
- Baseline: data set is on the cloud machine, passed to MXNet
- Telekine: data set is on a client, passed to MXNet instance
  - Telekine connects that instance to the remote GPU
  - As a result Telekine uses a consistent 533 Mb/s network bandwidth
### MXNet neural net training (Real WAN)

- Large dataset of images, processed in batches of size 64
- Baseline: data set is on **the cloud machine**, passed to MXNet
- Telekine: data set is on **a client**, passed to MXNet instance
  - Telekine connects that instance to the remote GPU
  - As a result Telekine uses a consistent 533 Mb/s network bandwidth

<table>
<thead>
<tr>
<th>Network</th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>DenseNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.23X</td>
<td>1.08X</td>
<td>1.22X</td>
</tr>
</tbody>
</table>
MXNet neural net training *(Real WAN)*

- Large dataset of images, processed in batches of size 64

Overheads are low because GPUs overlap the extra work with computation:
- E.g., CUs can keep processing while the DMA engine performs transfers
- Telekine connects that instance to the remote GPU
- As a result Telekine uses a consistent 533 Mb/s network bandwidth

<table>
<thead>
<tr>
<th></th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>DenseNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>1.23X</td>
<td>1.08X</td>
<td>1.22X</td>
</tr>
</tbody>
</table>
MXNet neural net training breakdown (Simulated WAN)

10ms RTT

(Real WAN)

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Add API Remoting</th>
<th>Add Encryption</th>
<th>Telekine</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>1.10</td>
<td>1.15</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td>InceptionV3</td>
<td>1.06</td>
<td>1.07</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>DenseNet</td>
<td>1.13</td>
<td>1.16</td>
<td>1.22</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>1.23X</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>1.08X</td>
</tr>
<tr>
<td>DenseNet</td>
<td>1.22X</td>
</tr>
</tbody>
</table>
### MXNet neural net training breakdown

For Simulated WAN with 10ms RTT:

<table>
<thead>
<tr>
<th>Network</th>
<th>Baseline</th>
<th>Add API Remoting</th>
<th>Add Encryption</th>
<th>Telekine</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>1.10</td>
<td>1.15</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td>InceptionV3</td>
<td>1.06</td>
<td>1.07</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>DenseNet</td>
<td>1.13</td>
<td>1.16</td>
<td>1.22</td>
<td></td>
</tr>
</tbody>
</table>

For Real WAN:

<table>
<thead>
<tr>
<th>Network</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>1.23X</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>1.08X</td>
</tr>
<tr>
<td>DenseNet</td>
<td>1.22X</td>
</tr>
</tbody>
</table>
MXNet neural net training breakdown (Simulated WAN)

10ms RTT

(Real WAN)

<table>
<thead>
<tr>
<th>Model</th>
<th>Baseline</th>
<th>Add API Remoting</th>
<th>Add Encryption</th>
<th>Telekine</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>1</td>
<td>1.10</td>
<td>1.15</td>
<td>1.19</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>1</td>
<td>1.06</td>
<td>1.07</td>
<td>1.10</td>
</tr>
<tr>
<td>DenseNet</td>
<td>1</td>
<td>1.13</td>
<td>1.16</td>
<td>1.22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>1.23X</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>1.08X</td>
</tr>
<tr>
<td>DenseNet</td>
<td>1.22X</td>
</tr>
</tbody>
</table>
MXNet neural net training breakdown (Simulated WAN)

10ms RTT

<table>
<thead>
<tr>
<th></th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>DenseNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slowdown</td>
<td>1.23X</td>
<td>1.08X</td>
<td>1.22X</td>
</tr>
</tbody>
</table>

(Real WAN)
Telekine: Secure Computing with Cloud GPUs

• Eliminates communication timing channels with data-oblivious streams

• Transparent to applications because it maintains GPU API semantics

• Has modest performance overheads for level of security provided
Telekine: Secure Computing with Cloud GPUs

- Eliminates communication timing channels with data-oblivious streams
- Transparent to applications because it maintains GPU API semantics
- Has modest performance overheads for level of security provided

Thanks!
Backup slides follow
MXNet training RTT sensitivity (Simulated WAN)

- RTT to cloud provider can vary
- The effect in performance depends on the workload

<table>
<thead>
<tr>
<th>RTT</th>
<th>ResNet50</th>
<th>InceptionV3</th>
<th>Densenet</th>
</tr>
</thead>
<tbody>
<tr>
<td>10ms</td>
<td>1.19X</td>
<td>1.10X</td>
<td>1.22X</td>
</tr>
<tr>
<td>20ms</td>
<td>1.29X</td>
<td>1.13X</td>
<td>1.37X</td>
</tr>
<tr>
<td>30ms</td>
<td>1.44X</td>
<td>1.16X</td>
<td>1.49X</td>
</tr>
<tr>
<td>40ms</td>
<td>1.53X</td>
<td>1.18X</td>
<td>1.66X</td>
</tr>
<tr>
<td>50ms</td>
<td>1.62X</td>
<td>1.30X</td>
<td>2.09X</td>
</tr>
</tbody>
</table>
Attack accuracy for batched inference

• GPU kernels operate on an entire batch
  - Cannot measure kernel execution time for individual images

• Task: correctly identify the class with the most images
  - Accuracy varies with how many more images there are (purity)
  - Batches of 32, four classes
  - Images selected from target class up to “Purity”
  - Batch filled out with images from other three classes

<table>
<thead>
<tr>
<th>Batch size</th>
<th>Purity</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100%</td>
<td>42%</td>
</tr>
<tr>
<td>32</td>
<td>25%</td>
<td>29%</td>
</tr>
<tr>
<td>32</td>
<td>80%</td>
<td>50%</td>
</tr>
<tr>
<td>32</td>
<td>100%</td>
<td>65%</td>
</tr>
</tbody>
</table>
Communication vs GPU work (Simulated WAN)

- Copy 16MB to the GPU
- Compute for x-axis seconds
- Copy 16MB from the GPU

Slowdown vs GPU Computation (in seconds, logscale)

Telekine