Visor: Private Video Analytics as a Cloud Service

Rishabh Poddar

Ganesh Ananthanarayanan, Srinath Setty, Stavros Volos, and Raluca Ada Popa

UC Berkeley and Microsoft Research
Machine learning as a service

- Image processing
- Speech recognition
- Video analytics
- …
Key application: Video analytics

Decisions based on objects in videos
Key application: Video analytics

Decisions based on objects in videos

• E.g. Raise an alert if a human at the main door
Key application: Video analytics

Decisions based on objects in videos

- E.g. Raise an alert if a human at the main door
- E.g. Change the traffic lights if less than 3 cars at the intersection
Video analytics pipeline

Pipeline of video processing modules

Client source

Cloud platform

Video decoding ➔ Background subtraction ➔ Bounding box detection ➔ Object cropping ➔ Objects

CNN classification

CPU

GPU
Video analytics pipeline

Pipeline of video processing modules

Cheap filters to discard useless frames
- Filter out moving objects
- CPU is under-utilized, unlike GPU

Client source
Cloud platform

Video decoding
Background subtraction
Bounding box detection
Object cropping

CPU

Objects

CNN classification

GPU
Video analytics pipeline

Pipeline of video processing modules

1. Red car
2. White van
3. Tree
4. …

Video decoding → Background subtraction → Bounding box detection → Object cropping

CNN classification

Results

Objects

Client source → Cloud platform
Problem: Privacy

Video stream is **proprietary** to clients
Related work: Privacy-preserving inference

Cryptographic approaches: e.g. SMPC, Homomorphic encryption
Related work: Privacy-preserving inference

Cryptographic approaches: e.g. SMPC, Homomorphic encryption

- High overhead
  - Can’t sustain video frame rate
Related work: Privacy-preserving inference

Cryptographic approaches: e.g. SMPC, Homomorphic encryption

- High overhead
  - Can’t sustain video frame rate
- Cannot be easily extended to video processing modules
Related work: Privacy-preserving inference

Cryptographic approaches: e.g. SMPC, Homomorphic encryption

- High overhead
  - Can’t sustain video frame rate
- Cannot be easily extended to video processing modules

Trusted hardware-based approaches (enclaves): e.g. Slalom, Chiron
Related work: Privacy-preserving inference

Cryptographic approaches: e.g. SMPC, Homomorphic encryption
  • High overhead
    — Can’t sustain video frame rate
  • Cannot be easily extended to video processing modules

Trusted hardware-based approaches (enclaves): e.g. Slalom, Chiron
  • Designed for CPU enclaves (using Intel SGX)
    — Can’t sustain video frame rate
Related work: Privacy-preserving inference

Cryptographic approaches: e.g. SMPC, Homomorphic encryption
- High overhead
  - Can’t sustain video frame rate
- Cannot be easily extended to video processing modules

Trusted hardware-based approaches (enclaves): e.g. Slalom, Chiron
- Designed for CPU enclaves (using Intel SGX)
  - Can’t sustain video frame rate
- Side-channel leakage — e.g. memory access patterns
Illustration of memory access pattern leakage

Example: Object detection

Original image (processed within SGX enclave)

Location of accessed pixels

Recovered image: Leaks shapes and positions of all objects
Key ideas

1. **Hybrid enclave** architecture (CPU + GPU) for secure computation

2. Remove memory side-channel leakage via **data-obliviousness**

   Redesign all components such that memory access patterns are **independent of data**
Hybrid trusted execution environment (TEE)

- **Intel SGX** (CPU TEE)
Hybrid trusted execution environment (TEE)

- **Intel SGX** (CPU TEE) + **Graviton** (GPU TEE)
Visor: Hybrid TEE for video analytics

- Ported to SGX using Graphene (extended to support required ioctl calls)
Visor: Hybrid TEE for video analytics

- Ported to SGX using Graphene (extended to support required ioctl calls)
Sources of leakage

- Ported to SGX using Graphene (extended to support required ioctl calls)
Sources of leakage

Encrypted traffic pattern
• E.g. leaks times of activity in video stream
Sources of leakage

SGX side-channel leakage
- Vast majority of known attacks on SGX exploit memory access pattern leakage

Cloud platform

Client source

Encrypted video stream

CPU enclave (Intel SGX)

Frame decoder
Object detection modules
Object buffer

GPU runtime

GPU enclave (Graviton)

CNN classifier

Results

Objects

25

2
Sources of leakage

CPU-GPU communication pattern
- **Number / size** of objects in frames

Client source

Encrypted video stream

Cloud platform

CPU enclave (Intel SGX)

Object detection modules

Frame decoder

Object buffer

GPU runtime

GPU enclave (Graviton)

CNN classifier

Results

Objects
Sources of leakage

- Frame decoder
- GPU runtime
- CNN classifier
- Cloud platform
- CPU enclave (Intel SGX)
- GPU enclave (Graviton)
- Client source
- Encrypted video stream
- Object detection modules
- Object buffer
- Results
- Objects
- Graviton side-channel leakage
  - Similar in principle to SGX side-channel leakage

Results
Sources of leakage

1. Cloud platform
2. CPU enclave (Intel SGX)
3. GPU enclave (Graviton)
4. CNN classifier

Client source
Encrypted video stream
Frame decoder
Object detection modules
Object buffer
GPU runtime
Results
Objects
Roadmap

1. Input traffic pattern
2. SGX memory side-channels
3. CPU-GPU communication pattern
4. Graviton memory side-channels
Roadmap

1. Input traffic pattern

2. SGX memory side-channels

3. CPU-GPU communication pattern

4. Graviton memory side-channels

Pad frames to an upper bound while enabling a bandwidth/performance trade-off
Roadmap

1. Input traffic pattern
2. SGX memory side-channels
3. CPU-GPU communication pattern
4. Graviton memory side-channels

Develop novel **data-oblivious** video processing algorithms
Roadmap

1. Input traffic pattern
2. SGX memory side-channels
3. CPU-GPU communication pattern
4. Graviton memory side-channels

Fixed communication schedule using dummy items, while minimizing GPU utilization
Roadmap

1. Input traffic pattern
2. SGX memory side-channels
3. CPU-GPU communication pattern
4. Graviton memory side-channels

Data-oblivious CNN implementation
Roadmap

1. Input traffic pattern
2. SGX memory side-channels
3. CPU-GPU communication pattern
4. Graviton memory side-channels
Roadmap

1. Input traffic pattern
2. SGX memory side-channels
3. CPU-GPU communication pattern
4. Graviton memory side-channels
Preventing SGX memory side-channels

Redesign algorithms so that memory access patterns are independent of secret data
Preventing SGX memory side-channels

Redesign algorithms so that memory access patterns are independent of secret data

- Identical set of operations per pixel
Preventing SGX memory side-channels

Redesign algorithms so that memory access patterns are independent of secret data

- Identical set of operations per pixel
- Design algorithms with structured memory accesses for efficiency
Preventing SGX memory side-channels

Redesign algorithms so that memory access patterns are independent of secret data

- Identical set of operations per pixel
- Design algorithms with structured memory accesses for efficiency

Implemented using a small set of oblivious primitives

- Assumption: Register-to-register operations are data-oblivious
- Wrappers around x86’s CMOV
Preventing SGX memory side-channels

Data-oblivious primitives

- **oselect** — conditional assignment

```c
if (cond) {
    y = x;
}
```

Can perform **dummy operations** by setting `cond` to false

```assembly
mov rcx, cond
mov rdx, x
mov rax, y
test rcx, rcx
cmovz rax, rdx
retn
```

**oselect(cond, x, y)**
Preventing SGX memory side-channels

Data-oblivious primitives\textsuperscript{1}

- \texttt{oselect} — conditional assignment
- \texttt{osort} — sorting using a bitonic network
- \texttt{oaccess} — array accesses

\textsuperscript{1}Ohrimenko et al.: “Oblivious multi-party machine learning on trusted processors” (Usenix Security ’16)
Illustration: Bounding box detection

Client source → Cloud platform → CPU → GPU

- Video decoding
- Background subtraction
- Bounding box detection
- Object cropping
- CNN classification

Objects
Illustration: Bounding box detection

Input: Binary image
Output: Bounding boxes of all objects
Illustration: Bounding box detection

**Input:** Binary image

**Output:** Bounding boxes of all objects

**Algorithm:** Border following

1. Scan image row-wise until a white pixel
**Illustration: Bounding box detection**

**Input:** Binary image

**Output:** Bounding boxes of all objects

**Algorithm:** Border following

1. Scan image row-wise until a white pixel
2. Examine neighbors and follow the border clock-wise to obtain edge contour
**Illustration: Bounding box detection**

**Input:** Binary image

**Output:** Bounding boxes of all objects

**Algorithm:** Border following

1. Scan image row-wise until a white pixel
2. Examine neighbors and follow the border clock-wise to obtain edge contour
3. Compute bounding box for each contour
**Illustration: Bounding box detection**

**Input:** Binary image

**Output:** Bounding boxes of all objects

**Algorithm:** Border following

1. Scan image row-wise until a white pixel
2. Examine neighbors and **follow the border** clock-wise to obtain edge contour
3. Compute bounding box for each contour

**Leakage:** Shape and location of each object
Oblivious Algorithm:

1. Scan image row-wise
2. For each white pixel:
   a. Assign smallest label of its already-scanned neighbors, or a new label.
   b. Update "parents" of all white neighbors to record that they are connected
   c. Update bounding box of assigned label
3. For each black pixel: same set of operations, but with dummy values (zeros)
4. Merge all connected labels

Illustration: Bounding box detection
Illustration: Bounding box detection

Oblivious Algorithm:
1. Scan image row-wise

   a. Assign smallest label of its already-scanned neighbors, or a new label.
   b. Update "parents" of all white neighbors to record that they are connected.
   c. Update bounding box of assigned label.

3. For each black pixel: same set of operations, but with dummy values (zeros).

4. Merge all connected labels.
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
**Oblivious Algorithm:**

1. Scan image row-wise

2. For each white pixel:
   a. Assign a **label** — smallest *already-scanned* neighbor, or a new label.
   b. Update “**parent**” of each neighboring label to record that they are **connected**

3. For each black pixel: same set of operations, but with dummy values (zeros)

4. Merge all connected labels
**Illustration: Bounding box detection**

**Oblivious Algorithm:**

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a **label** — smallest *already-scanned* neighbor, or a new label.
   b. Update “**parent**” of each neighboring label to record that they are *connected*
   c. Update label’s **bounding box**

---

Binary image

Parents

Bounding box coordinates
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a \textbf{label} — smallest \textit{already-scanned} neighbor, or a new label.
   b. Update “\textbf{parent}” of each neighboring label to record that they are \textit{connected}
   c. Update label’s \textbf{bounding box}

3. For each black pixel: same set of operations, but with \textbf{dummy} values (zeros)
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box

3. For each black pixel: same set of operations, but with dummy values (zeros)
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box

3. For each black pixel: same set of operations, but with dummy values (zeros)

4. Merge all connected labels
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a **label** — smallest *already-scanned* neighbor, or a new label.
   b. Update “**parent**” of each neighboring label to record that they are connected
   c. Update label’s **bounding box**

3. For each black pixel: same set of operations, but with **dummy** values (zeros)

---

**Binary image**

**Parents**

**Bounding box coordinates**
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box

3. For each black pixel: same set of operations, but with dummy values (zeros)

4. Merge all connected labels
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected.
   c. Update label’s bounding box.

3. For each black pixel: same set of operations, but with dummy values (zeros)

4. Merge all connected labels.
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.
2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box
3. For each black pixel: same set of operations, but with dummy values (zeros)
4. Merge all connected labels
**Illustration: Bounding box detection**

**Oblivious Algorithm:**

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a **label** — smallest *already-scanned* neighbor, or a new label.
   b. Update “**parent**” of each neighboring label to record that they are *connected*
   c. Update label’s **bounding box**

3. For each black pixel: same set of operations, but with **dummy** values (zeros)

---

**Binary image**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Parents**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bounding box coordinates**

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
<th>Top</th>
<th>Bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box

3. For each black pixel: same set of operations, but with dummy values (zeros)

4. Merge all connected labels
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box

3. For each black pixel: same set of operations, but with dummy values (zeros)

4. Merge all connected labels
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box

3. For each black pixel: same set of operations, but with dummy values (zeros)

   Binary image

   Parents

   Bounding box coordinates
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   
   a. Assign a **label** — smallest *already-scanned* neighbor, or a new label.

   b. Update “**parent**” of each neighboring label to record that they are connected

   c. Update label’s **bounding box**

3. For each black pixel: same set of operations, but with **dummy** values (zeros)

   *Binary image*

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

   *Parents*

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   *Bounding box coordinates*

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
<th>Top</th>
<th>Bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.
2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected
   c. Update label’s bounding box
3. For each black pixel: same set of operations, but with dummy values (zeros)
4. Merge all connected labels
Illustration: Bounding box detection

Oblivious Algorithm:

1. Scan image row-wise.

2. For each white pixel:
   a. Assign a label — smallest already-scanned neighbor, or a new label.
   b. Update “parent” of each neighboring label to record that they are connected.
   c. Update label’s bounding box.

3. For each black pixel: same set of operations, but with dummy values (zeros)

4. Merge all connected labels.

Binary image

Parents

Bounding box coordinates
Illustration: Bounding box detection

Enhancement via parallelization

Binary image
Illustration: Bounding box detection

Enhancement via parallelization

1. Divide the image into strips
Illustration: Bounding box detection

Enhancement via parallelization

1. Divide the image into **strips**
2. Detect bounding boxes per strip in **parallel**
Illustration: Bounding box detection

Enhancement via parallelization

1. Divide the image into **strips**
2. Detect bounding boxes per strip in **parallel**
3. **Merge** connected boxes across strip boundaries

Binary image
Performance highlights
Setup

Testbed

- Intel i7-8700K with 6 cores at 3.7 GHz
- NVIDIA GTX 780 GPU with 2304 CUDA cores operating at 863 MHz
Setup

Testbed

- Intel i7-8700K with 6 cores at 3.7 GHz
- NVIDIA GTX 780 GPU with 2304 CUDA cores operating at 863 MHz

Workload

- 4 real video streams of resolution 1280 x 720 and 1024 x 768
  - traffic cameras, security cameras
Setup

Testbed

- Intel i7-8700K with 6 cores at 3.7 GHz
- NVIDIA GTX 780 GPU with 2304 CUDA cores operating at 863 MHz

Workload

- 4 real video streams of resolution 1280 x 720 and 1024 x 768
  - traffic cameras, security cameras

Baseline

- Insecure video pipeline (without enclaves / obliviousness)
Overhead of side-channel protection

Net throughput (frames/s)

CNN model used in pipeline

- AlexNet
- ResNet-18
- ResNet-50
- VGG-16
- VGG-19

Baseline
Oblivious
Overhead of side-channel protection

Performance bottleneck shifts to the GPU with heavier models

<table>
<thead>
<tr>
<th>CNN model used in pipeline</th>
<th>Baseline</th>
<th>Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td><img src="http://example.com/alexnet.png" alt="" /></td>
<td><img src="http://example.com/oblivious.png" alt="" /></td>
</tr>
<tr>
<td>VGG-16</td>
<td><img src="http://example.com/vgg16.png" alt="" /></td>
<td><img src="http://example.com/oblivious.png" alt="" /></td>
</tr>
</tbody>
</table>
Overhead of side-channel protection

Performance bottleneck shifts to the GPU with heavier models, leaving more legroom for obliviousness on the CPU.
Overhead of enclaves

[Graph showing net latency (ms) for different frame resolutions and scenarios: Baseline, Baseline + enclaves, Oblivious + enclaves]
Overhead of enclaves

Frame resolution

Net latency (ms)

Baseline
Baseline + enclaves
Oblivious + enclaves
Overhead of enclaves

Limited CPU enclave memory size increases baseline latency for large frames

![Graph showing net latency (ms) vs frame resolution for different conditions: Baseline, Baseline + enclaves, Oblivious + enclaves. The graph indicates a 7.8x increase in latency for 1280 x 720 frame resolution compared to Baseline, and a 2.5x increase for 320 x 180 frame resolution.](image-url)
Overhead of enclaves

Limited CPU enclave memory size increases baseline latency for large frames
Summary

• Hybrid enclaves for running MLaaS applications such as video analytics pipelines

• Data-oblivious algorithms to remove memory side-channels

• Overhead of oblivious techniques limited to 2x—6x
Thanks!