Make Some ROOM for the Zeros
Data Sparsity in Secure Distributed Machine Learning

Phillipp Schooppmann, Adrià Gascón, Mariana Raykova, Benny Pinkas
June 20, 2019
Data Sparsity
Data Sparsity

- Netflix dataset: 480k users, 17k movies, but only 100M out of 8.5B potential ratings. $< 1.2\%$

- Genomics: 3.2B base pairs, but a typical genome differs only at 5M sites. $< 0.2\%$

- 20 Newsgroups dataset: 9k vectors, $10^5$ features, but only 100 non-zeros per vector. $< 0.1\%$
Sparse Storage Formats

Let $R$ be an arbitrary ring, $\mathbf{b} \in R^d$ be a vector, $\mathbf{A} \in R^{n \times d}$ a matrix.

- Sparse vector: $\text{SPARSE}(\mathbf{b}) := \{(i, b_i)\}_{b_i \neq 0}$

- Sparse matrix: $\text{SPARSE}(\mathbf{A}) := \left(\text{SPARSE}(\mathbf{a}_i)\right)_{i \in [n]}$

Related to the *Compressed Sparse Row (CSR)* or *Yale format* that is used in scientific computing libraries such as Eigen, SciPy, …
Secure Distributed Machine Learning
Throughout this talk: Two parties, semi-honest security
Two-Party Machine Learning

Throughout this talk: Two parties, semi-honest security
Throughout this talk: Two parties, semi-honest security
Two-Party Machine Learning

Throughout this talk: Two parties, semi-honest security
Building Block: Matrix-Vector Multiplication

Secure Computation

Choose random $[c]^A$, $[c]^B$, such that


Sparse Matrix-Vector Multiplication

😊 We don’t have to multiply elements if one of the factors is zero.
Sparse Matrix-Vector Multiplication

- We don’t have to multiply elements if one of the factors is zero.

- We can’t simply reveal which elements are zero.
Sparse Matrix-Vector Multiplication

😊 We don’t have to multiply elements if one of the factors is zero.

😢 We can’t simply reveal which elements are zero.

😊 In many settings, an upper bound on the number of non-zero elements is public.
Sparse Matrix-Vector Multiplication

- We don’t have to multiply elements if one of the factors is zero.
- We can’t simply reveal which elements are zero.
- In many settings, an upper bound on the number of non-zero elements is public.

Our Approach

1. Encode sparse vector in a Read-Only Oblivious Map (ROOM) data structure.
2. Implement matrix-vector multiplication as a batched oblivious map access.
Basic Primitive: ROOM
Read-Only Oblivious Maps

ROOM Protocol

Compute $w$ as

$$ w = \begin{cases} v & \text{if } (q, v) \in D \\ \bar{v} & \text{otherwise.} \end{cases} $$

Database $D$

- $(k_1, v_1)$
- $(k_2, v_2)$
- \vdots
- $(k_l, v_l)$

Default value $\bar{v}$

Query key $q$

Secret-shared value

$[w]^C$

$[w]^S$
Read-Only Oblivious Maps (2)

**Query keys**

$q_1$

$q_2$

\vdots

$q_k$

---

**Database $D$**

$(k_1, v_1)$

$(k_2, v_2)$

\vdots

$(k_l, v_l)$

**Default values**

$\bar{v}_1$

$\bar{v}_2$

\vdots

$\bar{v}_k$

---

**ROOM Protocol**

Compute $\mathbf{w}$ s.t. $\forall j \in [k]$: 

$$w_j = \begin{cases} 
    v & \text{if } (q_j, v) \in D \\
    \bar{v}_j & \text{otherwise.}
\end{cases}$$

---

**Secret-shared values**

$[\mathbf{w}]^C$

$[\mathbf{w}]^S$
### Related Functionalities

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Query privacy</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Database privacy</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sparse database</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Batched queries</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Shared Output</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
## Related Functionalities

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Query privacy</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Database privacy</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sparse database</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Batched queries</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Shared Output</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

*ROOM is shorter than *batched symmetric keyword PIR with shared output.* 😊
Read-Only Oblivious Maps (2)

Query keys
- \( q_1 \)
- \( q_2 \)
- \( \vdots \)
- \( q_k \)

Database \( D \)
- \( (k_1, v_1) \)
- \( (k_2, v_2) \)
- \( \vdots \)
- \( (k_l, v_l) \)

Default values
- \( \bar{v}_1 \)
- \( \bar{v}_2 \)
- \( \vdots \)
- \( \bar{v}_k \)

ROOM Protocol

Compute \( w \) s.t. \( \forall j \in [k] \):

\[
w_j = \begin{cases} 
    v & \text{if } (q_j, v) \in D \\
    \bar{v}_j & \text{otherwise.}
\end{cases}
\]
Building a ROOM

Naive approach: Ignore database sparsity.

1. Server extends database with dummy elements to span entire key domain:

$$\mathbf{x} = (\bot, \ldots, \bot, v_1, \bot, \ldots, \bot, v_2, \ldots)$$

index $k_1$  \hspace{1cm} index $k_2$
Building a ROOM

Naive approach: Ignore database sparsity.

1. Server extends database with dummy elements to span entire key domain:

   \[
   \mathbf{x} = (\perp, \ldots, \perp, v_1, \perp, \ldots, \perp, v_2, \ldots)
   \]

   \[\text{index } k_1 \quad \text{index } k_2\]

2. Server encrypts \(\mathbf{x}\) element-wise and sends it to client:

   \[
   \tilde{\mathbf{x}} = (\text{Enc}_K(x_1), \ldots, \text{Enc}_K(x_d))
   \]
Building a ROOM

Naive approach: Ignore database sparsity.

1. Server extends database with dummy elements to span entire key domain:

\[
x = (\bot, \ldots, \bot, v_1, \bot, \ldots, \bot, v_2, \ldots)
\]

\[
\uparrow \quad \text{index } k_1 \quad \uparrow \quad \text{index } k_2
\]

2. Server encrypts \(x\) element-wise and sends it to client:

\[
\tilde{x} = (\text{Enc}_K(x_1), \ldots, \text{Enc}_K(x_d))
\]

3. For each query \(q_i\), Client selects \(\tilde{x}_{q_i}\) and the parties perform an MPC with inputs \(\tilde{x}_{q_i}, K, \overline{v}_i\). The MPC
   a) Decrypts \(x_{q_i} = \text{Dec}_K(\tilde{x}_{q_i})\),
   b) Secret-shares \(x_{q_i}\) if \(x_{q_i} \neq \bot\), otherwise \(\overline{v}_i\).
Building a ROOM

Naive approach: Ignore database sparsity.

1. Server extends database with dummy elements to span entire key domain:

   \[ \mathbf{x} = (\bot, \ldots, \bot, v_1, \bot, \ldots, \bot, v_2, \ldots) \]

   \[ \text{index } k_1 \quad \text{index } k_2 \]

2. Server encrypts \( \mathbf{x} \) element-wise and sends it to client:

   \[ \tilde{\mathbf{x}} = (\text{Enc}_K(x_1), \ldots, \text{Enc}_K(x_d)) \]

3. For each query \( q_i \), Client selects \( \tilde{x}_{q_i} \) and the parties perform an MPC with inputs \( \tilde{x}_{q_i}, K, \tilde{\mathbf{v}}_i \). The MPC
   a) Decrypts \( x_{q_i} = \text{Dec}_K(\tilde{x}_{q_i}) \),
   b) Secret-shares \( x_{q_i} \) if \( x_{q_i} \neq \bot \), otherwise \( \tilde{v}_i \).

Communication linear in the key domain!
Building a ROOM

Naive approach: Ignore database sparsity.

1. Server extends database with dummy elements to span entire key domain:

   \[ \mathbf{x} = (\bot, \ldots, \bot, v_1, \bot, \ldots, \bot, v_2, \ldots) \]

   \[ \begin{array}{c}
     \text{index } k_1 \\
     \text{index } k_2
   \end{array} \]

2. Server encrypts \( \mathbf{x} \) element-wise and sends it to client:

   \[ \tilde{\mathbf{x}} = (\text{Enc}_K(x_1), \ldots, \text{Enc}_K(x_d)) \]

3. For each query \( q_i \), Client selects \( \tilde{x}_{q_i} \) and the parties perform an MPC with inputs \( \tilde{x}_{q_i}, K, \tilde{v}_i \). The MPC

   a) Decrypts \( x_{q_i} = \text{Dec}_K(\tilde{x}_{q_i}) \),
   
   b) Secret-shares \( x_{q_i} \) if \( x_{q_i} \neq \bot \), otherwise \( \tilde{v}_i \).

Communication linear in the key domain! 😞
Building a ROOM (2)

Idea: We don’t need to have an explicit representation of ⊥!
Building a ROOM (2)

Idea: We don’t need to have an explicit representation of ⊥!
Building a ROOM (2)

Idea: We don’t need to have an explicit representation of $\bot$!

1. Server pads and encrypts each value in the database:

$$\tilde{\mathbf{v}} = (\text{Enc}_K(v_1||0^s), \ldots, \text{Enc}_K(v_l||0^s))$$
Building a ROOM (2)

Idea: We don’t need to have an explicit representation of ⊥!

1. Server pads and encrypts each value in the database:

\[ \tilde{\mathbf{v}} = (\text{Enc}_K(v_1 || 0^s), \ldots, \text{Enc}_K(v_l || 0^s)) \]

2. Server interpolates and sends a polynomial \( P \) s.t. for all \( i \in [l] \)

\[ P(k_i) = \tilde{v}_i. \]
Building a ROOM (2)

Idea: We don’t need to have an explicit representation of `⊥`!

1. Server pads and encrypts each value in the database:

   \[ \tilde{v} = (\text{Enc}_K(v_1 || 0^s), \ldots, \text{Enc}_K(v_l || 0^s)) \]

2. Server interpolates and sends a polynomial \( P \) s.t. for all \( i \in [l] \)

   \[ P(k_i) = \tilde{v}_i. \]

3. For each query key \( q_i \), perform an MPC with inputs \( \tilde{x}_{q_i} = P(q_i), K, \tilde{v}_i \), that
   a) Decrypts \( x_{q_i} = \text{Dec}_K(\tilde{x}_{q_i}) \),
   b) Secret-shares \( v \) if \( x_{q_i} = (v || 0^s) \), otherwise \( \bar{v}_i \).
Sparse Inner Product from ROOM

Let $\text{SPARSE}(\mathbf{a}) := \{(i, a_i)\}_{a_i \neq 0}$, $\text{SPARSE}(\mathbf{b}) := \{(j, b_j)\}_{b_j \neq 0}$. 
Sparse Inner Product from ROOM

Let $\text{SPARSE}(a) := ((i, a_i))_{a_i \neq 0}$, $\text{SPARSE}(b) := ((j, b_j))_{b_j \neq 0}$.

Indices in $a$

\begin{align*}
i_1 \\
i_2 \\
\vdots \\
i_k
\end{align*}

\text{SPARSE}(b)

\begin{align*}
(j_1, b_{j_1}) \\
(j_2, b_{j_2}) \\
\vdots \\
(j_l, b_{j_l})
\end{align*}

Default values

\begin{align*}
0 \\
0 \\
\vdots \\
0
\end{align*}
Sparse Inner Product from ROOM

Let \( \text{SPARSE}(a) := ((i, a_i))_{a_i \neq 0} \), \( \text{SPARSE}(b) := ((j, b_j))_{b_j \neq 0} \).

Indices in \( a \)

\( i_1 \)

\( i_2 \)

\( \vdots \)

\( i_k \)

\( [b']^C \)

\( \text{SPARSE}(b) \)

Default values

\( (j_1, b_{j_1}) \)

\( (j_2, b_{j_2}) \)

\( \vdots \)

\( (j_l, b_{j_l}) \)

\( [b']^S \)

Now, \( b' = (b_i)_{a_i \neq 0} \). Let \( a' = (a_i)_{a_i \neq 0} \). Then \( ab = a'b' \).
APPLICATIONS
Logistic Regression

Logistic Regression: Time and Communication

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total Time</th>
<th></th>
<th>Communication</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SecureML</td>
<td>Ours</td>
<td>SecureML</td>
</tr>
<tr>
<td>Movies</td>
<td>6h29m28.37s</td>
<td>2h43m46.09s</td>
<td>4.8 TiB</td>
<td>187.42 GiB</td>
</tr>
<tr>
<td>Newsgroups</td>
<td>1h42m38.14s</td>
<td>42m37.68s</td>
<td>1.26 TiB</td>
<td>47.63 GiB</td>
</tr>
<tr>
<td>Languages, ngrams=1</td>
<td>5.9s</td>
<td></td>
<td>790.9 MiB</td>
<td>500.61 MiB</td>
</tr>
<tr>
<td>Languages, ngrams=2</td>
<td>1h3m7.12s</td>
<td>6m17.51s</td>
<td>797.85 GiB</td>
<td>3.69 GiB</td>
</tr>
</tbody>
</table>
$k$-Nearest Neighbors

$D = \text{such dataset}$

\[k\text{-NEAREST NEIGHBORS} \]

Find class $c_d$ most common among the $k$ nearest neighbors of $d$ in $D$.

**k-Nearest Neighbors: Time**

### Running Time

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Movies</th>
<th>Newsgroups</th>
<th>Languages, ngrams=1</th>
<th>Languages, ngrams=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Online time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>k-NN (LAN)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>** Dense**</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>** Basic-ROOM**</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>** Circuit-ROOM**</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>** Poly-ROOM**</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- To scale secure machine learning, we have to exploit characteristics in the *setting* and the *data*.

- We show that for *data sparsity*, using a dedicated data structure helps speed up multiple applications.
Conclusion

- To scale secure machine learning, we have to exploit characteristics in the setting and the data.

- We show that for data sparsity, using a dedicated data structure helps speed up multiple applications.

- Future directions:
  - Improve access times: LowMC, Cuckoo Hashing.
  - Adapt other primitives, e.g. Labeled PSI, flavors of PIR.
References I


References II


All icons made by Freepik from www.flaticon.com, licensed by CC BY 3.0.