SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search

Hao Chen (Microsoft)

Ilaria Chillotti (KU Leuven)

Yihe Dong (Microsoft)

Oxana Poburinnaya (University of Rochester)

Ilya Razenshteyn (Microsoft)

Sadegh Riazi (UCSD)

Secure computation

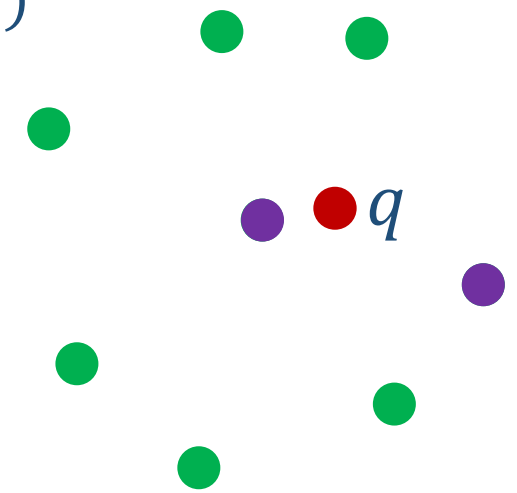
- Two or more parties want to perform some computation on their private data, and only reveal the output

- Example applications:
  - Private set intersection
  - Privacy-preserving machine learning
  - …

- Security notions: **passive/semi-honest**, covert, active/malicious

k-Nearest Neighbor Search (k-NNS)

- **Given:**
  - Dataset: n points P from a metric space $M = (X, D)$
- **Query:**
  - A point $q \in X$
- **Goal:**
  - Find k data points closest to the query q
- Can be used for **similarity search**



Secure k-NNS

- **Server** holds a dataset, **client** holds one or several queries
- Goal:
  - **Server** learns nothing
  - **Client** learns nothing about the dataset except (approximate) answers to the queries
- Accuracy vs. time, communication, number of rounds
- Focus on the **Euclidean distance**
- Applications: face recognition/biometric identification, searching medical data, etc.

Prior work

- **[Erkin, Franz, Guajardo, Katzenbeisser 2009] [Sadeghi, Schneider, Wehrenberg 2009] [Barni, Bianchi, Catalano, Raimondo, Labati, Failla, Fiore, Lazzeretti, Piuri, Scotti, Pivo 2010] [Evans, Huang, Katz, Malka 2011] [Demmler, Schneider, Zohner 2015] [Songhori, Hussain, Sadeghi, Koushanfar 2015] [Riazi, Chen, Shrivastava, Wallach, Koushanfar 2016] [Shaul, Feldman, Rus 2018] [Asharov, Halevi, Lindell, Rabin 2018] [Riazi, Javaheripi, Hussain, Koushanfar 2019]**
- All except one work implement **linear scan** securely

Possible solutions

- Linear scan
  - Query time: n distance computations
  - Too slow for massive datasets
- Sublinear-time algorithms
  - Approximate answers
  - **Might not work well with secure computation**

Our solution: hybrid protocols

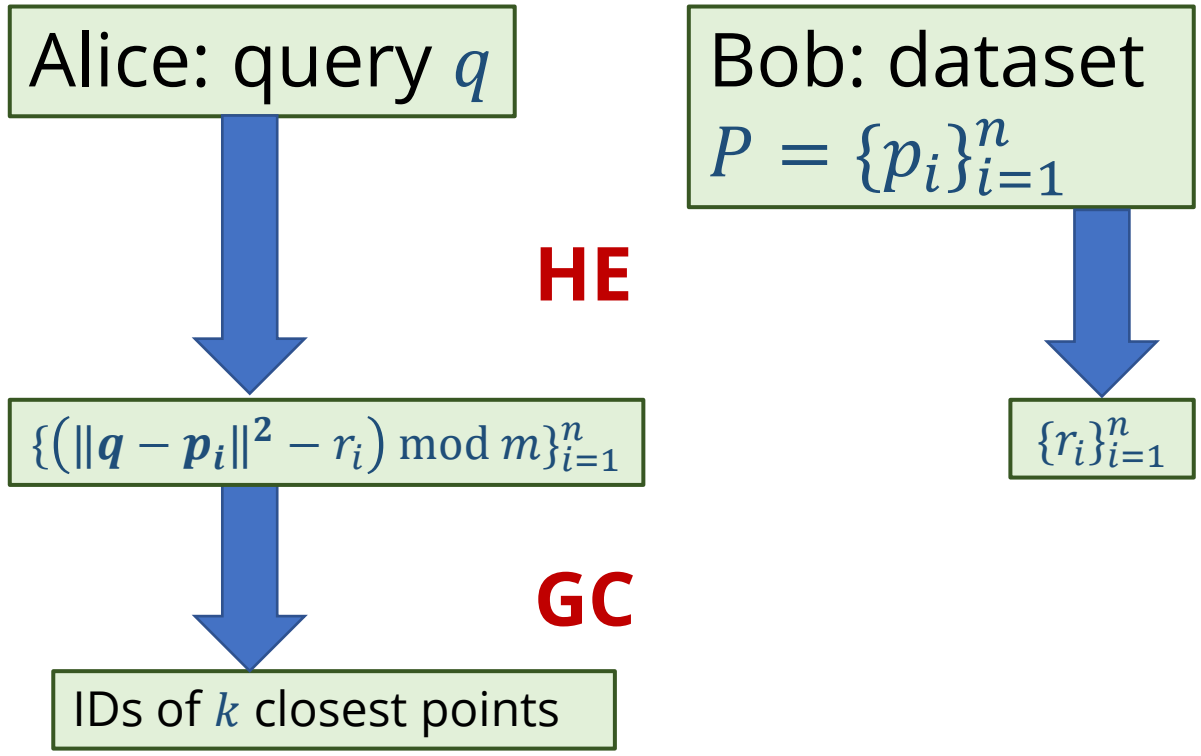
We propose two algorithms for secure **approximate** k-NNS:

- Optimized linear scan
- Cluster-based algorithm

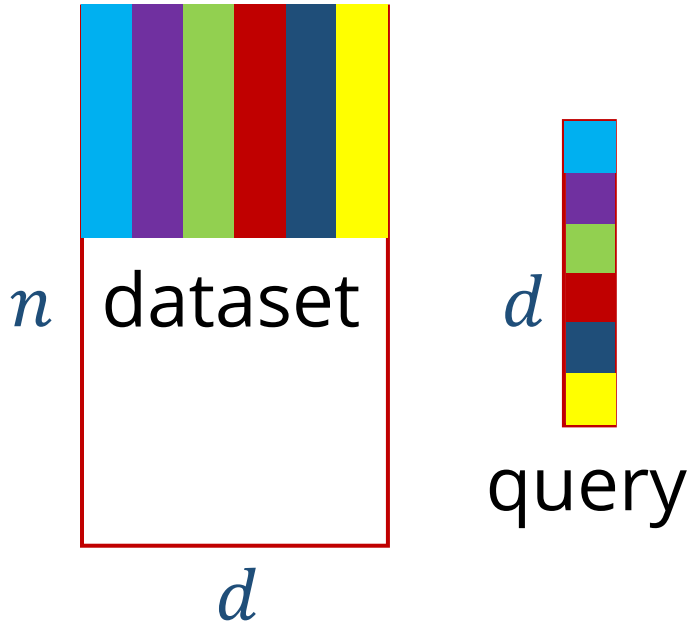
We implemented them securely using a combination of Homomorphic encryption (HE), Garbled circuits (GC), and Oblivious ROM (OROM)

Algorithm 1: Optimized linear scan

- **Linear scan:** compute distances + select k smallest
- Compute distance using HE and top-k selection using GC
- top-k selection becomes the bottleneck



Distance computation



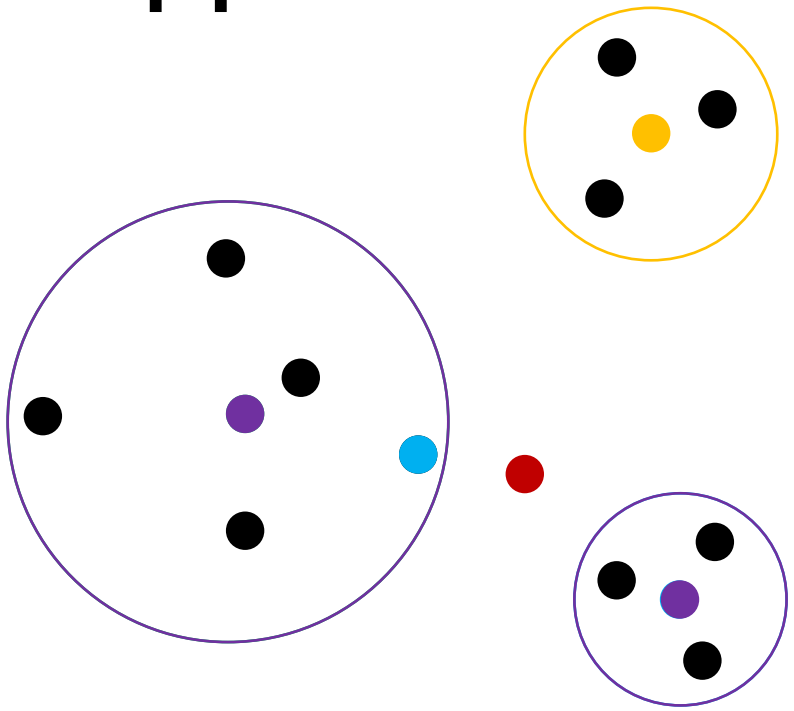
- $\|q - p_i\|^2 = \|q\|^2 + \|p_i\|^2 - 2\langle q, p_i \rangle$
- Enough to compute $-2\langle q, p_i \rangle$
- Use BFV scheme **[Brakerski 2012] [Fan, Vercauteren 2012]** as implemented in **Microsoft SEAL**
- Client encrypts the query, sends it to the server, server performs **additions and multiplications by a plaintext**
- Heavily vectorized

Randomized approximate top-k

- For n values with t bits each, naïve circuit need O(tnk) gates
- *Randomized* circuit with $O(t \cdot (n + \mathrm{poly}(k)))$ gates
  - For **every** input $x_1, x_2, \ldots, x_n$ circuit outputs k smallest numbers whp
- Partition into l **random** groups of size n/l
- Find minimum in each group: $O(tn)$ gates total
- Compute top-k among the minima: $O(tkl)$ gates
- Can choose $l = O(k^2/\delta)$ such that correct w.p. $1 - \delta$
- Overall, $O(n + k^3/\delta)$ comparisons, $O(t \cdot (n + k^3/\delta))$ gates

Algorithm 2: Clustering-based approach

- k-means clustering on the dataset
- Compute u centers closest to the query
- Return closest points from these clusters
- Run linear scan with retrieved points



Implementing Algorithm2 securely

- Compute distances to centers using **HE**
- Choose several closest centers using **GC**
- Retrieve (secret shares of) points from the corresponding clusters using **OROM** (one entry per cluster)
- Compute distances to the retrieved points using **HE**
- Choose closest points using **GC**

Oblivious Read-only Memory (ROM)

Securely returns $a[i]$ to Alice

- Linear communication complexity if done in GC
- Use **[Doerner, Shelat 2017]:** $O(\log n)$ communication
  - ~ $\log n$ rounds

Experiments

	Algorithm	Per-client Preprocessing	OT Phase	Query			
				Total	Distances	Top-k	ORAM
SIFT	Linear scan	None	1.83 s / 21.6 s 894 MB	33.3 s / 139 s 4.51 GB	19.8 s / 25.6 s 98.7 MB	13.5 s / 111 s 4.41 GB	None
	Clustering	12.6 s / 24.7 s 484 MB	0.346 s / 4.34 s 156 MB	8.06 s / 59.7 s 1.77 GB	2.21 s / 3.67 s 56.7 MB	1.96 s / 18.0 s 645 MB	3.85 s / 38.1 s 1.06 GB
Deep 1B-1M	Linear scan	None	1.85 s / 20.6 s 894 MB	28.4 s / 133 s 4.50 GB	14.9 s / 20.6 s 86.1 MB	13.5 s / 112 s 4.41 GB	None
	Clustering	11.0 s / 20.6 s 407 MB	0.323 s / 4.09 s 144 MB	6.95 s / 47.8 s 1.58 GB	1.66 s / 3.13 s 44.1 MB	1.93 s / 16.6 s 620 MB	3.37 s / 27.9 s 920 MB
Deep 1B-10M	Linear scan	None	20.0 s / 232 s 9.78 GB	375 s / 1490 s 47.9 GB	202 s / 201 s 518 MB	173 s / 1280 s 47.4 GB	None
	Clustering	86.0 s / 167 s 3.71 GB	1.04 s / 13.4 s 541 MB	30.1 s / 181 s 5.53 GB	6.27 s / 10.2 s 59.4 MB	7.22 s / 61.0 s 2.35 GB	16.5 s / 107 s 3.12 GB
Amazon	Linear scan	None	1.99 s / 23.3 s 960 MB	22.9 s / 133 s 4.85 GB	8.27 s / 14.0 s 70.0 MB	14.6 s / 118 s 4.78 GB	None
	Clustering	7.27 s / 13.4 s 247 MB	0.273 s / 3.17 s 108 MB	4.54 s / 35.2 s 1.12 GB	0.68 s / 2.31 s 24.4 MB	1.64 s / 13.8 s 528 MB	2.22 s / 18.8 s 617 MB

Table 1: Evaluation of SANNS in a single-thread mode. Preprocessing is done once per client, OT phase is done once per query. In each cell, timings are given for fast and slow networks, respectively.

Experiments

	Algorithm	Threads								Speed-up
		1	2	4	8	16	32	64	72	
SIFT	Linear scan	33.3 s	23.2 s	13.4 s	8.04 s	4.78 s	4.25 s	**3.96 s**	4.14 s	8.4
		139 s	76.4 s	46.9 s	32.5 s	25.7 s	22.1 s	**20.9 s**	21.3 s	6.7
	Clustering	8.06 s	4.84 s	3.16 s	2.18 s	1.65 s	1.55 s	**1.44 s**	1.47 s	5.6
		59.7 s	35.2 s	23.6 s	24.4 s	20.1 s	14.2 s	**11.1 s**	12.1 s	5.4
Deep 1B-1M	Linear scan	28.4 s	19.9 s	11.4 s	7.39 s	4.53 s	**3.94 s**	**3.94 s**	4.05 s	7.2
		133 s	75.5 s	44.5 s	31.9 s	24.5 s	22.0 s	22.5 s	**21.1 s**	6.3
	Clustering	6.95 s	4.20 s	2.62 s	2.03 s	1.52 s	1.43 s	**1.37 s**	1.39 s	5.1
		47.8 s	28.5 s	22.0 s	23.0 s	18.4 s	14.7 s	**11.0 s**	11.5 s	4.3
Deep 1B-10M	Linear scan	375 s	234 s	118 s	81.8 s	65.8 s	55.0 s	**53.1 s**	58.5 s*	7.1
		1490 s	800 s	480 s	343 s	266 s	231 s	**214 s**	216 s*	7.0
	Clustering	30.1 s	18.0 s	10.8 s	7.21 s	4.85 s	4.58 s	**4.23 s**	4.25 s	7.1
		181 s	97.5 s	60.0 s	54.5 s	48.1 s	37.2 s	30.3 s	**28.4 s**	6.4
Amazon	Linear scan	22.9 s	15.4 s	10.1 s	6.66 s	4.14 s	3.73 s	3.78 s	**3.64 s**	6.3
		133 s	73.1 s	46.1 s	33.8 s	26.2 s	24.1 s	22.0 s	**21.7 s**	6.1
	Clustering	4.54 s	2.66 s	1.87 s	1.40 s	1.17 s	1.15 s	**1.12 s**	1.16 s	4.1
		35.2 s	21.4 s	14.9 s	16.8 s	14.2 s	11.5 s	10.8 s	**9.19 s**	3.8

Table 2: Evaluation of SANNS query algorithms in the multi-thread mode. Each cell contains timings for fast and slow networks. Optimal settings are marked in bold. For the numbers marked with an asterisk, we take the *median* of the running times over several runs, since with small probability (approximately $20-30\%$), memory swapping starts due to exhaustion of all the available RAM, which affects the running times dramatically (by a factor of $\approx 2\times$).

Conclusion

- We improved the performance of secure k nearest neighbors
- (Much) faster secure implementation of the linear scan
  - Small Boolean circuit for top-k (two new constructions)
- First secure implementation of a **sublinear time** algorithm
  - **New algorithm** tailored to secure computation
- A number of optimizations to HE, GC, OROM
  - Dramatically improved concrete efficiency
- Can find 10-NN on 10M 96-dimensional vectors with accuracy 0.9 in **under 6 seconds**
  - Up to **31x faster** than (optimally implemented) prior work

Thank you!

Contact: Hao Chen (haoche@microsoft.com)