



Privacy-Aware Document Search on the Cloud

293S, 2020, T. Yang

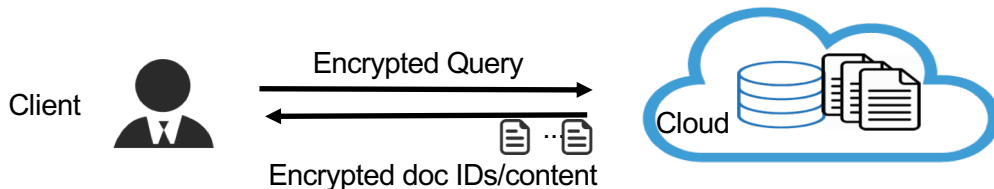
Outline

- **Privacy-aware search**
 - Motivation & problem definition
 - Searchable encryption & known attacks
- **Privacy-aware ranking**
 - Tree-ensemble based ranking
 - Ranking with neural signals

No homework questions on this subject

Privacy-aware Search: Motivations

- Client uploads private data collections to the cloud such as gmail, source code, enterprise documents
- Privacy-aware search does not want server to know the hosted content and index, and what is searched



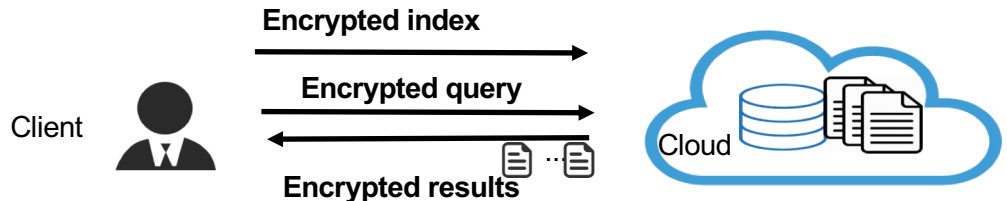
- Server is **honest-but-curious**: correctly executes protocols but observes/infers private information
 - **Plain text leakage occurs** due to accidents, misconfiguration/use, or attacks.
 - “Dropbox Security Bug Made Passwords Optional For Four Hours”. June 2011

Searchable Encryption: Workflow

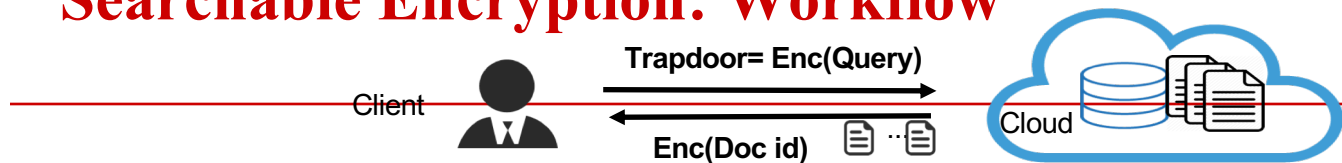
Three entities: data owner, search user and cloud server.
Data owner and a user are the same for this presentation

- [SongS&P2000]: Song, et al. , "Practical techniques for searches on encrypted data." IEEE Symposium on Security and Privacy, 2000.
- Offline preprocessing

- [Client] **KeyGen** (a security parameter):
 - Generate security keys
 - Output: Private key k_I for index, k_D for documents.
- [Client] **BuildIndex** (k_I , k_D , **D**):
 - Build encrypted index
 - Output: Secure index **I** and encrypted documents **C**.



Searchable Encryption: Workflow



- [Client] **KeyGen** (a security parameter):
- [Client] **BuildIndex** (k_I , k_D , **D**):
- [Client] **TokenGen** (k_I , q): Online query processing
 - Client converts query q to encrypted tokens using key k_I
 - Output: Query tokens **t** (**encrypted query, or called trapdoor**)
- [Server] **Search** (I , **t**):
 - Search encrypted index with query token(s)
 - Output: encrypted matched document ID **C**
- [Client] **Decryt** (k_D , **C**):
 - Client decrypts document IDs
 - Output: Matched documents.

What are possible privacy attacks?

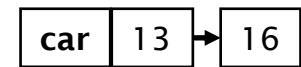
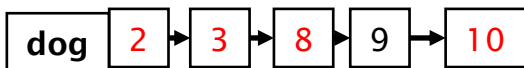
- Documents and queries are encrypted. The doc and term IDs are also random numbers in the index with no meaning
 - Why there exist privacy attacks
- **Leakage-Abuse Attacks:**
 - Recover plaintext of queries or documents
 - IKK attack
 - Islam, et al. "Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation." NDSS. Vol. 20. 2012.
 - Launchable when a server knows co-occurrence probability of English words + some other info

What does a server know in IKK: Example

- Knows co-occurrence matrix M , element M_{ij} is the probability of the i -th keyword w_i and the j -th keyword w_j appearing in the same document.

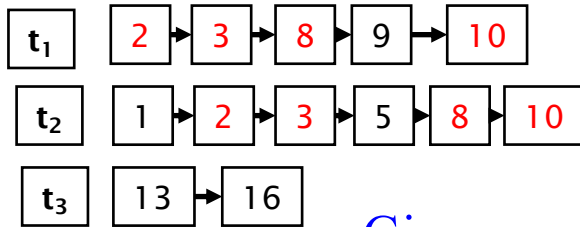
How a server can estimate the elements of M_{ij} without knowing plaintext?

	dog	cat	car	trunk
dog	1.0	0.8	0.3	0.2
cat	-	1.0	0.1	0.6
car	-	-	1.0	0.9
trunk	-	-	-	1.0



$M_{\text{dog,cat}} = ?$ 4/10 assuming 10 documents in total

Compute co-occurrence probability from index access patterns



Given a query token t , access pattern $R(t)$ is a list of all document IDs matched and returned

- The index has 10 documents
 $R(t_1) = \{2, 3, 8, 9, 10\}$
 $R(t_2) = \{1, 2, 3, 5, 8, 10\}$
- $Cooccurrence[t_1, t_2] = 4/10 = 0.4$

- In general, a server can estimate an element of concurrence matrix after processing two single-word queries

How can a server attack?

Co-occurrence matrix M: Adversary knows English word for M, but does not know corresponding tokens.

Co-occurrence matrix M': The adversary does not know English word of each column. It estimates M' by index access.

Try a mapping between M' and M: **t1=dog, t2=cat, t3=car**
mapping error : $(0.8-0.3)^2 + (0.2-0.3)^2 + (0.9-0.1)^2$

M'

	t1	t2	t3	
t1	1.0	0.3	0.2	
t2	-	1.0	0.9	
t3	-	-	1.0	

M

	dog	cat	car	trunk
dog	1.0	0.8	0.3	0.2
cat	-	1.0	0.1	0.6
car	-	-	1.0	0.9
trunk	-	-	-	1.0



try: t1=dog, t2=car, t3=trunk

IKK Assumptions: What a server knows

- Each query uses one single word
- Know the plaintext of m potential searchable keywords and their $m \times m$ co-occurrence matrix M
 - Can be approximated from similar datasets. E.g. Wikipedia.
 - But do not know their term IDs in the index
- Observe online processing of q single-word queries
 - Observe index access patterns: all documents IDs that match each query token
- Know in advance the text-to-token mapping for a small numbers of words, e.g. Inject 150-word public doc or email
- Objective: Recover plaintext of these q words

Steps to conduct IKK attack

Step 1: Calculate co-occurrence matrix M' for these q query tokens based on the index access pattern:

- Each column corresponds to a token. The adversary does not know the plaintext of its corresponding word
- $M'_{i,j}$ is estimated as intersection size of posting lists of words w_i and w_j , divided by #documents in the dataset

Step 2: Map M' to a submatrix of M :

Adversary knows keywords for M , but does not know corresponding tokens.

Use simulated annealing to map tokens to English words and minimize the squared error of pair mapping:

$$\sum (\text{cooccurrence probability mapping difference})^2$$

IKK attack limitation and extension

Counter attack = IKK + known term frequency

by Cash et al. "Leakage-abuse attacks against searchable encryption."
2015 ACM CCS. <https://eprint.iacr.org/2016/718.pdf>

- IKK attack is not scalable for a large dataset (>5000 words).

- Counter attacks requires that a server knows the document frequency of terms (e.g. posting list length). Does not require known keyword-to-token mapping
- More scalable

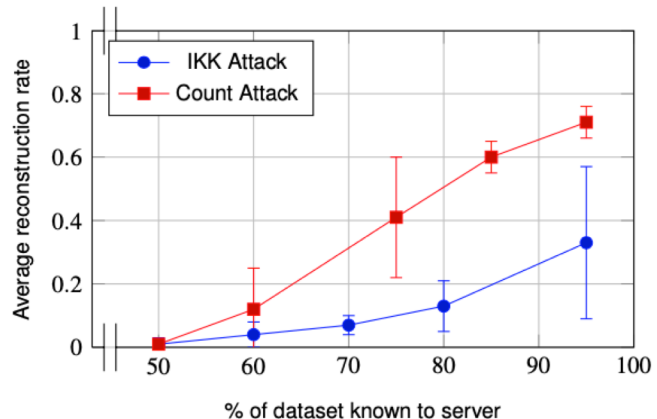


Figure 6: Average reconstruction rate when server has partial knowledge of true document set for the Enron dataset with 500 keywords and 150 queried uniformly. 5% of queries are known by IKK. The count attack has no known queries. Error bars show one standard deviation.

Outline

- **Privacy-aware/secure search**
 - Motivation & problem definition
 - Searchable encryption & known attacks
- **Privacy-aware ranking**
 - Tree-ensemble based ranking
 - Ranking with neural signals



Challenges in Privacy-Aware Rank Computation

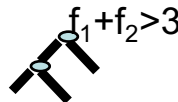
- **Ranking requires arithmetic computation and comparison**
 - Feature composition: e.g. TF-IDF, BM25, word distance.
 - Linear/nonlinear rank computation and comparison.
- **Computation and comparability of encrypted features**
 - Compose $E(f_1^d + f_2^d)$ from $E(f_1^d)$ and $E(f_2^d)$ securely?
 - Compare $E(f_1^{d1} + f_2^{d1})$ and $E(f_1^{d2} + f_2^{d2})$ securely?
 - Fully Homomorphic encryption [Gentry STOC09]:
Slow

Privacy-aware Ranking with Tree Ensembles on the Cloud [Ji SIGIR'18]

- Full server-side ranking with encoded feature values for privacy protection
- Tree-based learning ensembles (GBRT, LamdaMART, Random Forest) still need to compose input features with arithmetic operations
 - E.g. BM25 involves additions
 - feature leakage during server tree computation
- **Proposed scheme:**
 - Simplify required arithmetic operations
 - Query-length-specific training
 - Hide feature values and tree thresholds with comparison-preserved mapping

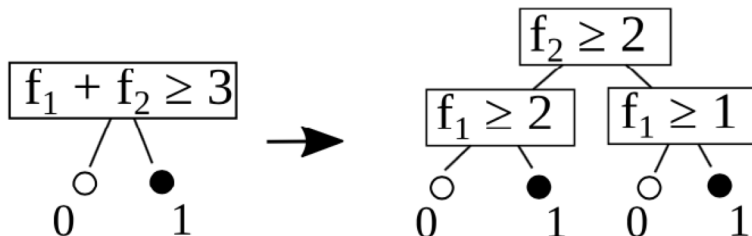
Idea 1: Simplified feature composition

- More server-side operation types supported → more difficult to preserve privacy



- Strategy:

- Restrict type of arithmetic operations in feature compositions.
- Only support min/max based composition from raw features



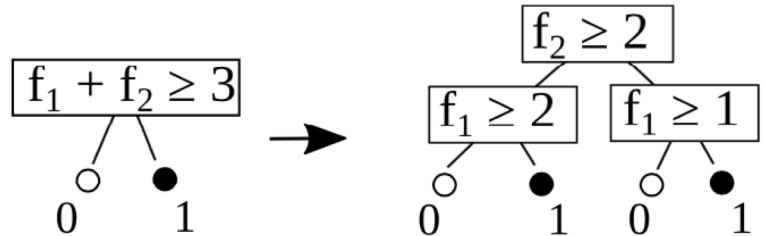
- For **BM25**, use individual raw features (avoid addition)
- For **proximity**, use feature values of terms which are word pairs or n-gram terms

Ranking without addition: Example

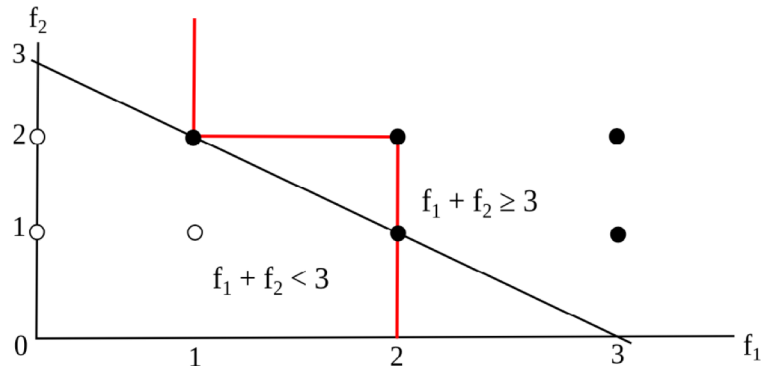
- Query: CD rate
- Document: d="CD rate is 0.02 provided by CD bank"
- **Feature vector for a traditional tree algorithm:**
 - d= [Average TF of unigram query words, Average TF of bigram query words]
$$d=[(2+1)/2, 1]$$
- **Feature vector for ranking** with no addition involved:
= [TF of CD, TF of rate, TF of CD-rate] = [2, 1, 1]

Property of tree transformation by removing sum feature operators

Using raw features only for common operators has no training loss degradation in terms of squared error or entropy-based information gain



- The new tree can separate white and black circles as accurate as the old tree



Idea 2: Encoding with comparison-preserved mapping

- **Objective:** Hide document feature values and tree thresholds for better privacy
- **Option 1: OPM**
 - Order preserved mapping [Boldyreva et al. Crypto11]
 - $v_1 > v_2 \Leftrightarrow \text{OPM}(v_1) > \text{OPM}(v_2)$
 - $v_1 = v_2 \Leftrightarrow \text{OPM}(v_1) = \text{OPM}(v_2)$
- **Option 2: CPM** (Comparison preserved mapping)
Feature value/threshold mapping only preserves correctness of decision tree branching
Leak less: $v_1 \geq v_2 \Leftrightarrow \text{CPM}(v_1) \geq \text{CPM}(v_2)$



Example of comparison-preserved mapping (CPM)

A decision tree has 3 thresholds

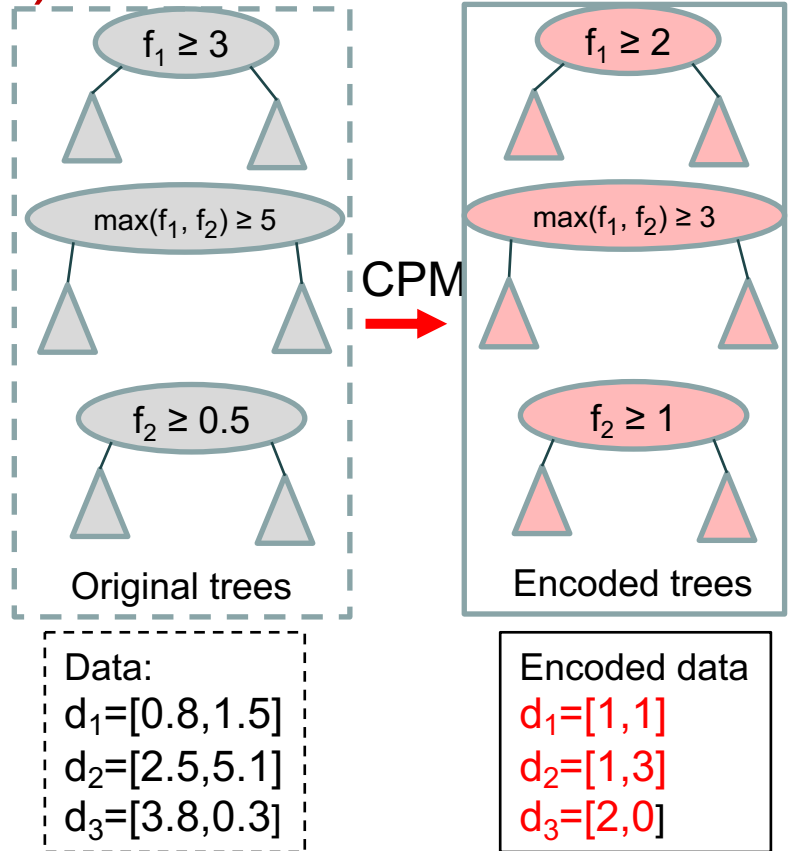
$[0.5, 3, 5]$

\downarrow
 $[1, 2, 3]$

The dataset has 6 feature values

$[0.3, 0.8, 1.5, 2.5, 3.8, 5.1]$

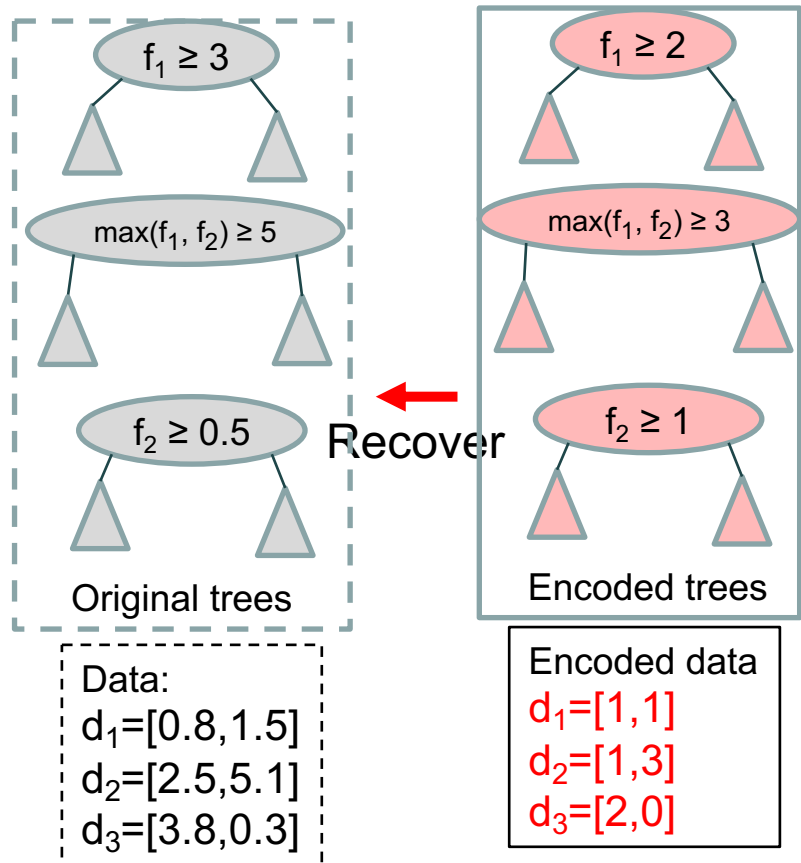
\downarrow
 $[0, 1, 1, 1, 2, 3]$



What information is protected

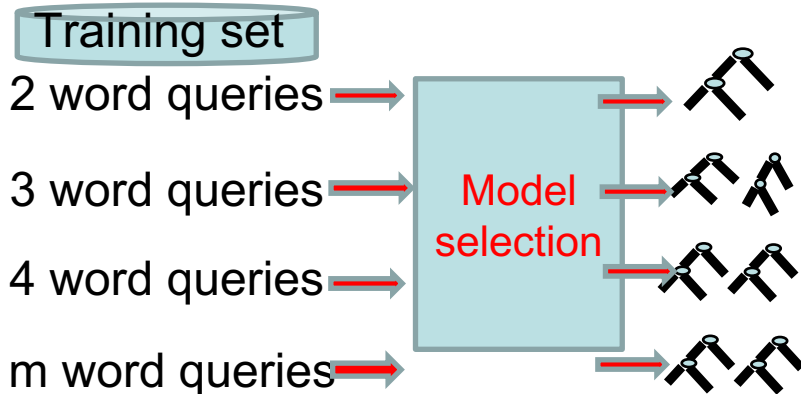
A server cannot well recover original values of feature values, their difference, and their ratios.

- If it can do within an error bound, it has to distinguish the original data from an infinite number of possible datasets, which is unlikely.



Ideal 3: Query-length-specific training

- Number of raw features is query-dependent.
- Query-length specific training with hybrid tree ensemble



Allow a different algorithm to be used for a different query length with a different combination of raw/composite features

Evaluation with CPM-based tree ensembles

- **Assume features can be safely fetched for matched documents**
- **Evaluation objective:** Can tree ensembles with CPM using raw and min/max compositions perform competitively?
- **TREC Datasets**

Query length	1	2	3	4	5
Robust04, 0.5M	11	70	140	25	4
Robust05, 1M	1	19	24	5	1
ClubeWeb09-12, 50M	64	70	52	14	0
ClubeWeb, MQ09, 50M	98	294	232	53	9

Relevance of CPM trees with restricted features

Compared to existing methods with no Restriction
5-fold validation NDCG@20 results

Collections	λ - MART	GBRT	Random Forest	CPM
Robust04	0.3936	0.4025	0.4114	0.3975 (-3.3%)
Robust05	0.2765	0.2778	0.2945	0.2928 (-0.6%)
ClueWeb09-12	0.2235	0.1906	0.2100	0.2160 (-3.4%)
ClueWeb09, MQ09	0.2603	0.2419	0.2395	0.2573 (-1.2%)

CPM is close to the best constantly with small degradation

Relevance with different query lengths

NDCG@20 of ClueWeb09, MQ09. Features include raw individual BM25 for title/body, word-pair distance with min/max composition, PageRank, and Wikipedia indicator

Q-length	λ -MART	GBRT	Random Forest	CPM
2	0.2712	0.2457	0.2612	0.2712 (0%)
3	0.2683	0.2185	0.2284	0.2767 (+3.1%)
4	0.2280	0.2296	0.2369	0.2296 (-3%)
5	0.0913	0.0843	0.0388	0.0913 (0%)

CPM gives the more stable results than others by selecting the best configuration with query-length specific optimization.

Privacy-aware Neural Ranking [Shao et al. SIGIR 2019]

- **Interaction-based Neural Ranking**

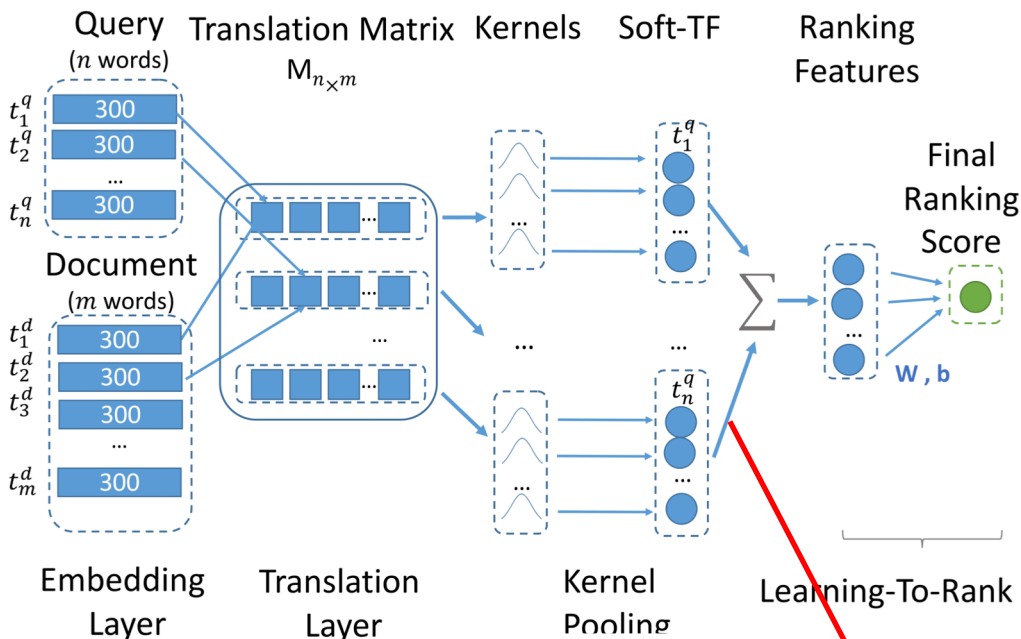
DRMM, KNRM,
Conv-KNRM

- $\text{RankingScore} = f(\text{Ker}(\mathbf{q} \otimes \mathbf{d}))$
 - \mathbf{q}, \mathbf{d} : sequences of embedding vectors for query and document
 - \otimes : interaction matrix
 - Ker : kernel computation
 - f : neural network computation
- Interaction \otimes outputs a similarity matrix M containing vector similarity for all pairs of one query term vector and document term vector

- Kernel computation:
$$K_k(M_i) = \sum_{j=1}^n \exp\left(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2}\right)$$

Neural Ranking in KNRM

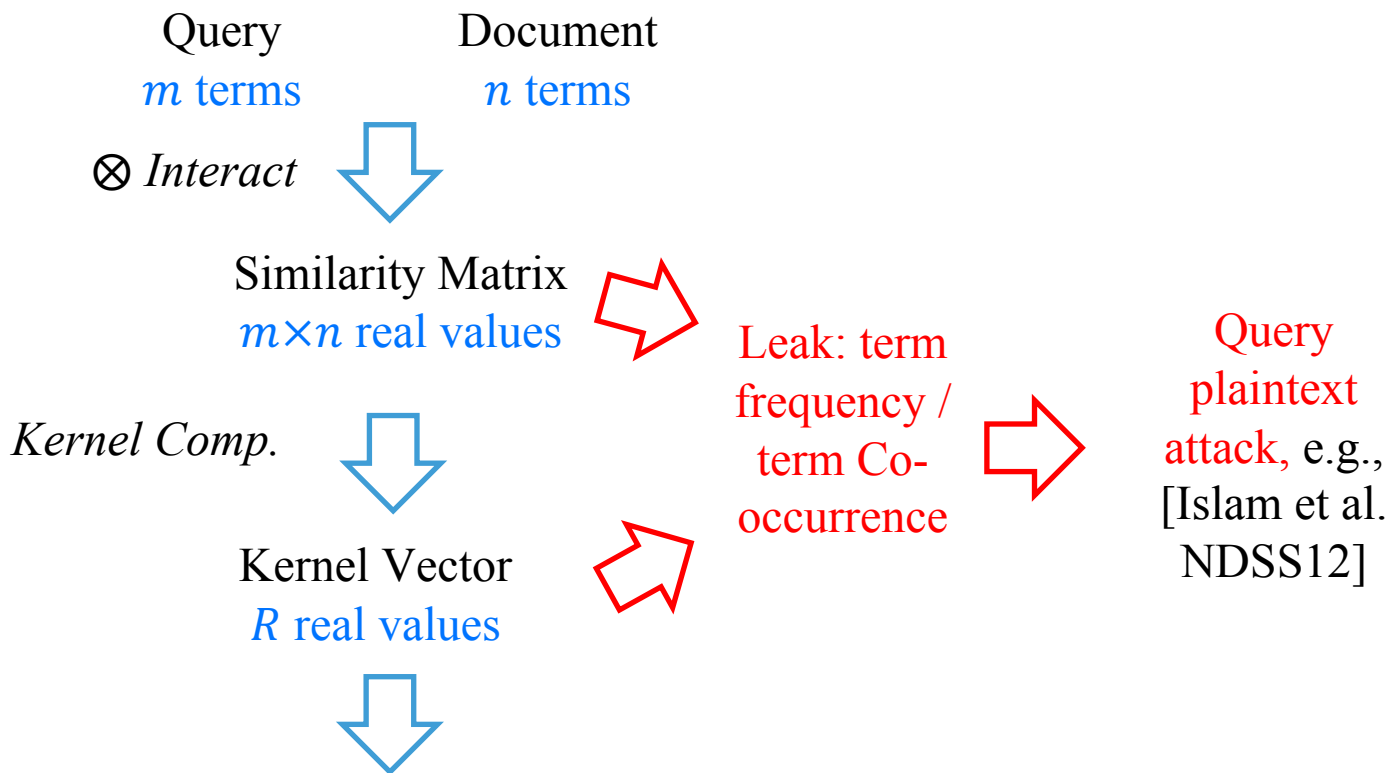
- Translation matrix is a cosine similarity of a query term and a document term



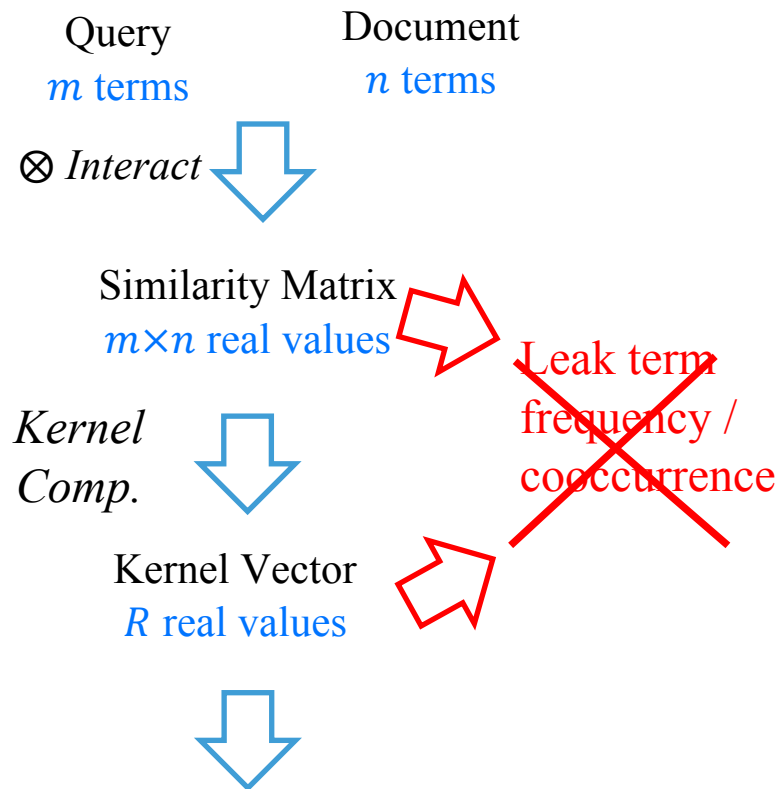
$$(\sum_{t \in q} \log K_1(t, d), \dots, \sum_{t \in q} \log K_R(t, d))^T.$$

$$K_j(t, d) = \sum_{w \in d} \exp\left(-\frac{(\langle t, w \rangle - \mu_j)^c}{2\sigma_j^2}\right).$$

Leakage in Interaction-based Neural Ranking



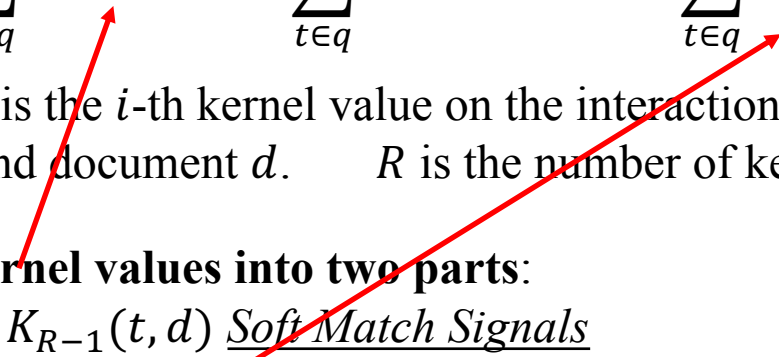
Proposed Solution for Private Neural Ranking



1. Leakage in computing kernel vectors?
Pre-computing them.
2. Avoid leakage in kernel vectors?
Partial replacement with private tree ensembles
3. Too much storage cost in storing precomputed kernel vectors?
Closed soft match map

How Kernel Values Leak Term Frequency

Final kernel vector for query q and document d :

$$\left\{ \sum_{t \in q} \log K_1(t, d), \sum_{t \in q} \log K_2(t, d), \dots, \sum_{t \in q} \log K_R(t, d) \right\}$$


$K_i(t, d)$ is the i -th kernel value on the interaction of query term t and document d . R is the number of kernels.

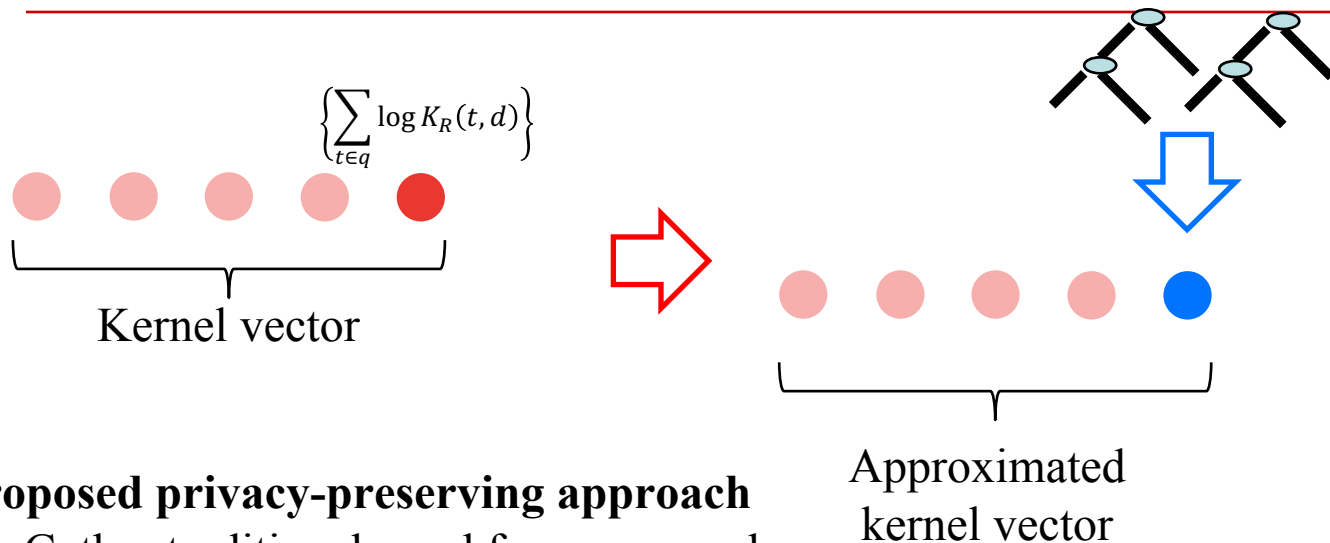
Decompose kernel values into two parts:

- $K_1(t, d), \dots, K_{R-1}(t, d)$ Soft Match Signals
- $K_R(t, d)$ Exact Match Signal

Our analysis: Term frequency of t can be well approximated by $K_R(t, d)$.

Solution for privacy-preserving: Replace the exact match signal with the private tree-based ranking signal.

How to Approximate Exact Match Signal $K_R(t, d)$



Proposed privacy-preserving approach

1. Gather traditional word frequency and proximity features
2. Use a query-length-specific learning-to-ranking tree ensemble to compute a rank score
3. Use a private tree-based model [Ji et al., SIGIR18] to encrypt features and tree thresholds

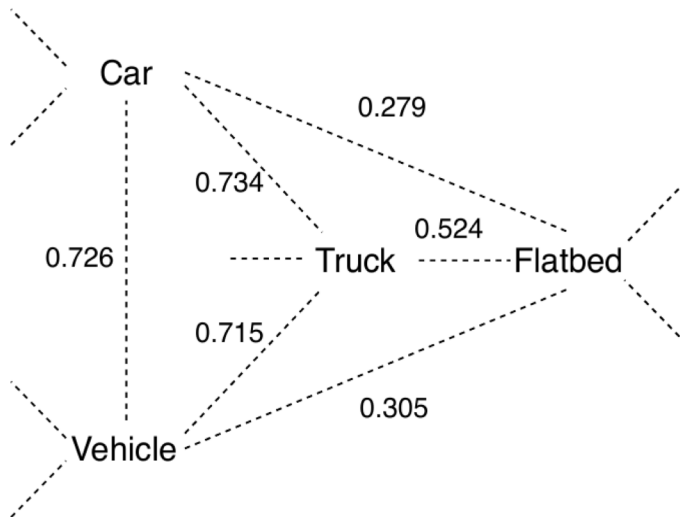
Storage Cost Reduction with Soft Index

Store precomputed kernel vectors for all term-doc pairs
→ **Huge storage cost.** 16TB for 1M documents and 200K vocabulary.

Soft indexing solution: soft match map

- Index contains posting (term, doc) only if this term is semantically related to this document.
- For every term t in a dataset, form a closure for term t above a similarity threshold
- Given term t in document d , we find other terms t' in the cluster of t . Then the index contains posting (t', d)

Clustering for Soft Match Map



Clustering with fixed threshold:

C1: {Car, Truck, Vehicle, Flatbed, ...}

Adaptive clustering:

C1: {Car, Truck, Vehicle},
C2: {FlatBed, ...}

Fixed threshold: Can create a super large closure.

Adaptive clustering: Only keep top similar terms with different threshold for different clusters, which avoids massive storage cost.

Privacy Property of Closed Soft Match Map

Objective: Given a closed soft match map, a server adversary cannot learn term frequency/co-occurrence of the dataset.

Property Sketch: Given a dataset D , with N key-value pairs in a closed soft match map of D , and closure size $\geq p$, there exist at least $(2^p - 1)^N$ different datasets D' such that their soft match maps have the same key, and values that are ε -statistically indistinguishable

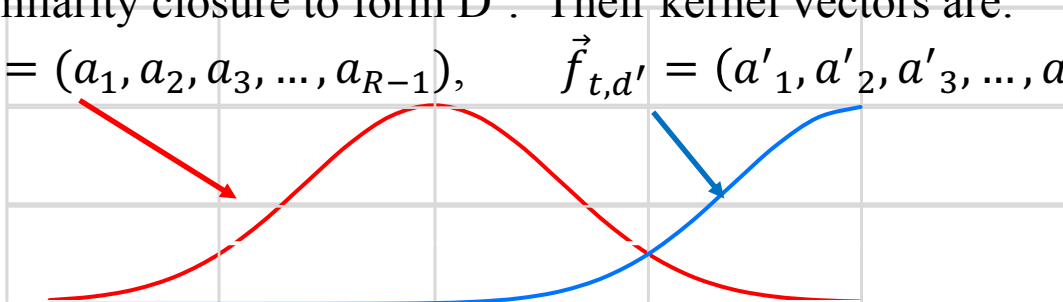
Takeaway: These $(2^p - 1)^N$ different datasets have different term frequencies and co-occurrences, while their soft match maps are very similar (*kernel value indistinguishability*).

Thus, the cloud server is unlikely to recover the correct dataset.

Kernel Value Indistinguishability: Definition

Given dataset D , transform words in document d in D as d' by using a similarity closure to form D' . Their kernel vectors are:

$$\vec{f}_{t,d} = (a_1, a_2, a_3, \dots, a_{R-1}), \quad \vec{f}_{t,d'} = (a'_1, a'_2, a'_3, \dots, a'_{R-1}).$$



- **ϵ -statistically indistinguishable kernel values:**

The statistical difference between D and $D' \leq \epsilon$

- An adversary can successfully differentiate D and D' with probability at most $\frac{1}{2} + \epsilon$.

- **Takeaway**

$\downarrow \epsilon \xrightarrow{\text{yields}} \downarrow \text{Prob}(\text{successfully differentiate between } D \text{ and } D')$

Obfuscating kernel values for privacy protection

$$\left\{ \sum_{t \in q} \log K_j(t, d) \right\}$$



For the j -th soft kernel value in the kernel value vector, it is **obfuscated** as:

$$a_j = \begin{cases} \lfloor \log_r K_j(t, d) \rfloor, & \text{if } K_j(t, d) > 1, \\ 1, & \text{otherwise,} \end{cases}$$

t is a term, d is a document, and $K_j(t, d)$ is the output from the j -th kernel function.

Trade-off between privacy and ranking accuracy:

\uparrow Obfuscation value $r \xrightarrow{\text{yields}} \downarrow$ *Statistical Distance*

$\xrightarrow{\text{yields}} \uparrow$ Privacy $\xrightarrow{\text{yields}} \downarrow$ **Effectiveness of soft match signals**

Evaluation on Approx. Exact Match Signal

	ClueWeb09-Cat-B			Robuts04		
Model	NDCG@1	NDCG@3	NDCG@10	NDCG@1	NDCG@3	NDCG@10
LambdaMART	0.2893	0.2828	0.2827	0.5181	0.4610	0.4044
DRMM	0.2586	0.2659	0.2634	0.5049	0.4872	0.4528
KNRM	0.2663	0.2739	0.2681	0.4983	0.4812	0.4527
C-KNRM	0.3155	0.3124	0.3085	0.5373	0.4875	0.4586
C-KNRM*	0.2884	0.2927	0.2870	0.5007	0.4702	0.4510
C-KNRM*/T	0.3175	0.3122	0.3218	0.5404	0.5006	0.4657

C-KNRM is CONV-KNRM [Dai et al. WSDM18]

C-KNRM* is a version of CONV-KNRM without bigram-bigram interaction

C-KNRM*/T is C-KNRM* while using a LambdaMART tree ensemble to replace the exact match signal of kernel vectors.

Takeaway: Tree signal intergration for neural kernel vectors perform well and even boost ranking performance.

Effectiveness of Kernel Value Obfuscation

	ClueWeb09-Cat-B			Robuts04		
Model	NDCG@1	NDCG@3	NDCG@10	NDCG@1	NDCG@3	NDCG@10
C-KNRM	0.3155	0.3124	0.3085	0.5373	0.4875	0.4586
C-KNRM*	0.2884	0.2927	0.2870	0.5007	0.4702	0.4510
C-KNRM*/TO No Obfuscation	0.3175	0.3122	0.3218	0.5404	0.5006	0.4657
C-KNRM*/TO r = 5	0.3178	0.3067	0.3100	0.5306	0.4987	0.4613
C-KNRM*/TO r = 10	0.3121	0.3097	0.3100	0.5221	0.4980	0.4623

C-KNRM*/TO is C-KNRM* while using the tree-approximated kernel vectors and kernel value obfuscation

Takeaway: Kernel value obfuscation results in small degradation ($\sim 1.6\%$) on ranking performance, when $r = 10$.

Summary

- **Secure search with searchable encryption**
 - Knowing con-occurrence and document frequency of terms can aid privacy-abuse attacks
- **Privacy-aware ranking**
 - Tree-ensembles: Transform features and avoid additions. Comparison-preserved mapping
 - Ranking with neural signals: Replace the exact match kernel with privacy-aware trees.
Precompute kernel vectors