Tiptoe: Private Web Search

Ioannis Notaris (cs32200038)

National and Kapodistrian University of Athens

January 10, 2024

Abstract

Tiptoe is a private web search engine ensuring client privacy over hundreds of millions of documents.

- Privacy based on cryptography alone
- Utilizes semantic embeddings for private full-text search
- Implements private nearest-neighbor search with linearly homomorphic encryption

Introduction

- Web searches reveal personal data to search engines.
- Privacy risks
- IP anonymizers like Tor may not fully protect user privacy.
- Common algorithms and data structures require search engines to see the user's query.
- Tiptoe addresses privacy concerns (it learns nothing about user queries).
- Strong privacy guarantee solely based on cryptographic assumptions.

Introduction (contd.)

- Semantic embeddings for document selection and ranking inspired by non-private search engines.
- Reduces private web search to private nearest-neighbor search, supporting various media types.
- The Tiptoe servers perform most computations in a preprocessing step, minimizing per-query computation.
- Documents are clustered by topic to reduce communication costs.
- Tiptoe's prototype on a 45-server cluster achieves 2.7 seconds end-to-end latency for private web search.

Goals and Limitations

Tiptoe's Goal:

- Tiptoe aims to be a search engine that learns nothing about users' search queries.

Implications of Query Privacy

- Tiptoe's query privacy ensures all aspects of its behavior are independent of the client's query, up to cryptographic assumptions.
- Servers never see the client's query in plaintext.
- The search engine does not learn the set of search results sent back to the client.

Non-Goals and Limitations

- Tiptoe does not hide the time or the frequency of queries.
- Does not protect information about a client's web-browsing behavior post-query.
- Does not guarantee service availability or result correctness against malicious servers.
- Embedding-based search returns semantic matches, introducing machine learning limitations.

Tiptoe Design

Key Principles for Query Privacy:

- Every client protocol message is encrypted with a secret key known only to the client.
- Message flow and packet sizes are independent of the client's secret, query string or the servers' behavior.
- Servers compute answers directly on encrypted data without decryption.

Design Ideas

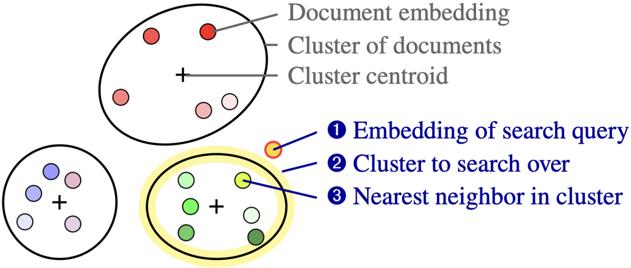
Core Challenge:

- Balancing expressive queries, hiding query contents, and searching over a vast number of documents in seconds.
- Techniques to address this challenge:
  - Embedding-based search
  - Private nearest-neighbor search with fast linearly homomorphic encryption
  - Clustering to reduce communication

Embedding-based Search

- Tiptoe uses semantic embeddings for document representation.
- Embeddings are small, allowing efficient linear scan by servers.
- Supports expressive queries without special machinery for keyword analysis.
- Compatible with various embedding models (e.g., transformer models).

Tiptoe's semantic search with embeddings



Document embedding

Cluster of documents

Cluster centroid

❶ Embedding of search query

❷ Cluster to search over

❸ Nearest neighbor in cluster

Private Nearest-neighbor Search

- Tiptoe achieves private nearest-neighbor search with fast linearly homomorphic encryption.
- Client sends an encrypted query embedding; servers compute inner product with every document.
- Linearly homomorphic encryption simplifies computation and ensures good performance.
- Client is required to download ciphertexts for inner-product scores.

Clustering to Reduce Communication

- Tiptoe employs clustering to reduce client-server traffic ($\sqrt{N}$ clusters for N documents).
- Documents with similar embeddings grouped into clusters.
- Client downloads cluster “centroids” ahead of time, reducing communication cost.
- Privacy maintained through cryptographic protocols despite clustering.

Tiptoe Architecture

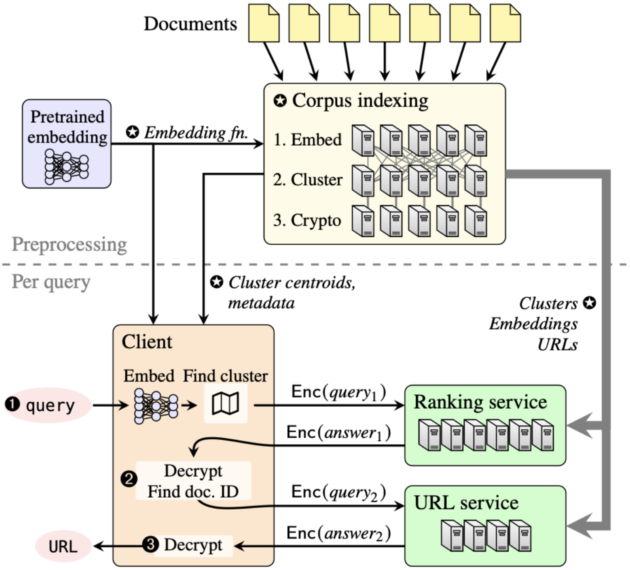
Components:

- Data-loading batch jobs
- Client
- Ranking service
- URL service

Steps in Data-loading Batch Jobs:

1. Embed
2. Cluster
3. Preprocess cryptographic operations

The Tiptoe system architecture



Tiptoe Architecture (contd.)

Tiptoe Search Process:

1. Embed query
2. Rank documents using the ranking service
3. Fetch URLs using the URL service

Handling Updates to the Corpus:

- Continuous updates supported by running new or changed documents through the embedding function.
- Updated cluster centroids and metadata published to clients.

Tiptoe's Private Ranking Service

Objective:

- Allow the client to find the IDs of the most relevant documents to its query.
- Implementation based on a new private nearest-neighbor search protocol.

Approximate Nearest Neighbors:

- The protocol provides approximate nearest neighbors, considering semantic embeddings.
- No formal correctness guarantees, aligning with the nature of semantic embeddings.

Private Nearest-Neighbor Protocol

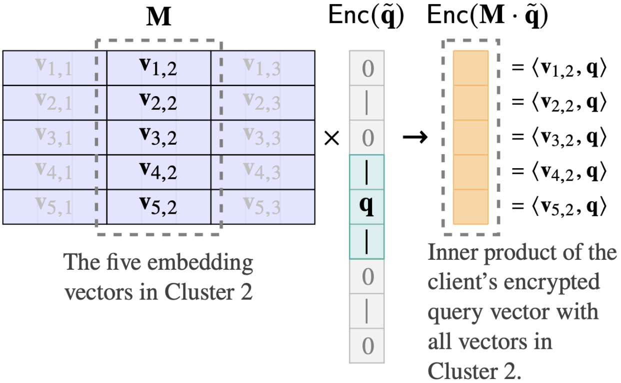
Protocol Steps:

1. **Client Query Preparation:** Client identifies the cluster nearest to its query embedding, prepares the query vector $\tilde{\mathbf{q}}$, encrypts it, $\mathrm{ct} = \mathrm{Enc}(\tilde{\mathbf{q}})$ and sends it to the ranking service.
2. **Ranking-Service Computation:** Ranking service computes matrix-vector product under encryption.

$$\mathrm{ct}' = \mathbf{M} \cdot \mathrm{Enc}(\tilde{\mathbf{q}}) = \mathrm{Enc}(\mathbf{M} \cdot \tilde{\mathbf{q}})$$

3. **Client Decryption:** Client decrypts and identifies nearest neighbors.

Private nearest-neighbor computation


$\mathbf{M}$
Enc($\tilde{\mathbf{q}}$)
Enc($\mathbf{M} \cdot \tilde{\mathbf{q}}$)
$\mathbf{v}_{1,1}$ $\mathbf{v}_{1,2}$ $\mathbf{v}_{1,3}$
$\mathbf{v}_{2,1}$ $\mathbf{v}_{2,2}$ $\mathbf{v}_{2,3}$
$\mathbf{v}_{3,1}$ $\mathbf{v}_{3,2}$ $\mathbf{v}_{3,3}$
$\mathbf{v}_{4,1}$ $\mathbf{v}_{4,2}$ $\mathbf{v}_{4,3}$
$\mathbf{v}_{5,1}$ $\mathbf{v}_{5,2}$ $\mathbf{v}_{5,3}$
×
0
0
$\mathbf{q}$
0
0
→
$= \langle \mathbf{v}_{1,2}, \mathbf{q} \rangle$
$= \langle \mathbf{v}_{2,2}, \mathbf{q} \rangle$
$= \langle \mathbf{v}_{3,2}, \mathbf{q} \rangle$
$= \langle \mathbf{v}_{4,2}, \mathbf{q} \rangle$
$= \langle \mathbf{v}_{5,2}, \mathbf{q} \rangle$

The five embedding vectors in Cluster 2

Inner product of the client's encrypted query vector with all vectors in Cluster 2.

The matrices in the private nearest-neighbor protocol with $C = 3$ clusters of 5 vectors each. The client searches within cluster 2, uploads an encrypted query vector $\tilde{\mathbf{q}}$ and receives the encrypted inner-product scores for all documents in the cluster.

Private Nearest-Neighbor Protocol (contd.)

Analysis:

- Security: Server sees only the encryption of the client's augmented query vector.
- Correctness: Provides approximate nearest neighbors based on semantic embeddings.
- Communication Cost: Scales roughly as $d\sqrt{N}$ $(C \approx \sqrt{N})$.
- Performance: Computes a matrix-vector product.

Implementation Considerations

Representing Real-Valued Embeddings:

- Tiptoe uses linearly homomorphic encryption, which operates on integers modulo p.
- Real numbers represented as integers modulo p using fixed-precision representation.

Scaling Out to Many Physical Machines:

- Sharding matrix M across multiple servers for reduced latency and fault tolerance.
- Front-end coordinator and worker machines handle query computation.

Tiptoe's URL Service

Functionality:

- Fetch metadata for the IDs from the previous step.
- Metadata includes document URLs, potentially also web-page titles, summaries, or image captions.
- Tiptoe client fetches metadata for the top 100 search results by default.

Private Information Retrieval

Protocol:

- Tiptoe uses single-server private information retrieval protocols (very similar to private nearest-neighbor search).
- SimplePIR with additional optimizations.

How it Works:

- SimplePIR client builds a vector indicating the record it wants to retrieve.
- Vector is encrypted using linearly homomorphic encryption and sent to the server.
- Server computes on fixed-length ciphertexts to retrieve the requested record.

Optimizations for Data Retrieval

Chunk Size and Compression:

- SimplePIR serves data in relatively large chunks (about 40 KiB).
- Tiptoe compresses batches of URLs to fit into these chunks efficiently.
    1. Compressing Batches of URLs
    2. Grouping URLs by Content

Cryptographic Optimizations

Overview of Optimizations:

1. Preprocessing to reduce per-query computation.
2. Compression of ciphertexts using a second layer of homomorphic encryption.
3. Reducing Latency with Query Tokens.

Preprocessing to reduce per-query computation

- The Matrix $\mathbf{M}$ does not depend on the clients' query.
- It can be preprocessed to speed up the matrix vector product under encryption.
- It can be reused until the corpus changes.

Compression of Ciphertexts

Motivation:

- Large ciphertexts from homomorphic encryption $\mathbf{C} = \mathrm{Enc}(\mathbf{M} \cdot \tilde{\mathbf{q}})$.
- Roughly 4096 times larger than corresponding plaintext.
- To decrypt it computes the matrix-vector product $\mathbf{y} = \mathbf{C} \cdot \mathbf{s}$ where $\mathbf{s}$ is the secret key.

Optimization Technique:

- Inspired by "bootstrapping" in fully homomorphic encryption.
- Outsourcing decryption work to the server using a second linearly homomorphic encryption scheme (Enc2).
  1. Client encrypts the secret key $\mathbf{s}$ using Enc2.
  2. Server computes $\mathbf{C} \cdot \mathrm{Enc}_2(\mathbf{s}) = \mathrm{Enc}_2(\mathbf{C} \cdot \mathbf{s}) = \mathrm{Enc}_2(\mathbf{y})$.

Reducing Latency with Query Tokens

Optimization Approach:

- Pushing client-to-server communication to a per-query preprocessing step.
- Reduces perceived latency for clients.

Query Tokens:

- $\mathbf{C}$ depends 99.9% on the document corpus.
- Client sends an encrypted secret key $(\mathrm{Enc}_2(\mathbf{s}))$ to the server.
- Downloads the product $\mathbf{C} \cdot \mathrm{Enc}_2(\mathbf{s})$ before deciding on the query string.
- Query tokens are fetched in advance and used for one search query each.

Tiptoe Prototype Overview

Source Code:

- Available at `github.com/ahenzinger/tiptoe`.
- Consists of approximately 5,200 lines of code.

Components:

- Tiptoe client and services (3,700 lines of Go and Python).
- Batch jobs (1,500 lines of Python).
- Cluster management (1,000 lines of Python).

Embedding Models

Text Search:

- Model: msmarco-distilbert-base-tas-b text.
- Embedding Vectors: Dimension 768.
- First 512 tokens used for each document.

Text-to-Image Search:

- Model: CLIP embedding function [107].
- Embedding Vectors: Dimension 512.
- Modification for plain image search requires minimal code changes.

Dimensionality Reduction and Clustering

Dimensionality Reduction:

- Principal component analysis (PCA) on document embeddings.
- Reduced dimension to 192 for text search and 384 for image search.

Clustering:

- Faiss library used for grouping documents.
- Clusters consist of approximately 50,000 documents.

Optimizations in Clustering

Multiple Clusters Assignment:

- Tiptoe assigns documents to multiple clusters if close to boundaries.
- 20% of documents assigned to two clusters.
- 80% assigned to a single cluster.
- Results in 1.2x overhead in server computation and communication.

Evaluation Questions

- How good are Tiptoe's text-search results? (§8.2)
- What is the performance and cost of Tiptoe? (§8.3)
- How do Tiptoe's costs compare to those of other private-search systems? (§8.4)
- How well does Tiptoe scale to larger corpuses? (§8.5)
- To what extent do our optimizations reduce Tiptoe's search costs and affect its search quality? (§8.6)

Search Quality

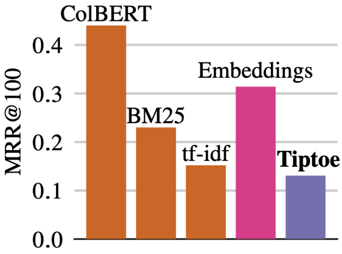
Comparison of Search Algorithms on MS MARCO:

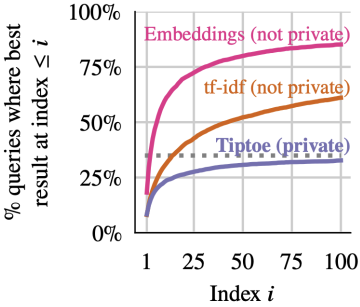
- ColBERT (deep-learning-based), BM25 (keyword-based), tf-idf, exhaustive embedding search, and Tiptoe.
- MRR@100 score comparison.

Search Results Distribution:

- Tiptoe's distribution compared to tf-idf and exhaustive embedding search.
- Tiptoe correctly identifies and searches within the cluster containing the best result on approximately 35% of queries.

Search Quality Results





End-to-End Performance

Text Search:

- Cost: $0.003 per query.
- Latency: 2.7 seconds.

Image Search:

- Cost: $0.008 per query.
- Latency: 3.5 seconds.

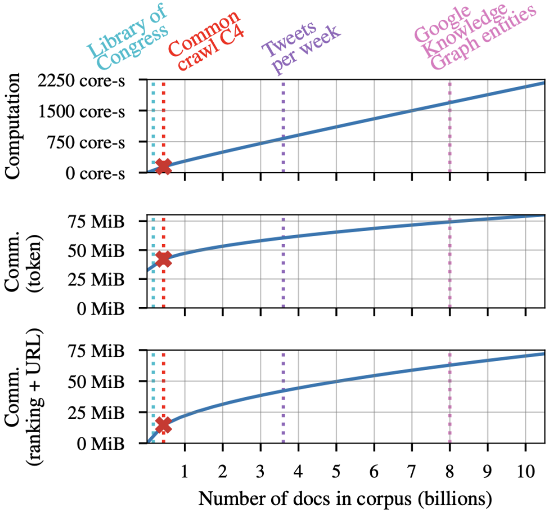
Baseline Comparisons:

- Client-side search index (48 GiB storage).
- Coeus query-scoring costs (vs. Tiptoe's query costs).

Search Quality Results

Cost metrics:	Client storage (GiB)	Comm. (MiB/query)	Server comp. (core-s/query)	End-to-end latency (s)	AWS cost ($/query)
Wikipedia search over 5 million docs					
Coeus query-scoring [5]	0	50	12 900	2.8	0.059
Text search over 360 million docs					
Client-side Tiptoe index	48	0	0	0	0
Tiptoe	0.3	$42^\diamond + 15$	145	2.7	0.003
Image search over 400 million docs					
Client-side Tiptoe index	98	0	0	0	0
Tiptoe	0.7	$50^\diamond + 21$	339	3.5	0.008

Scaling to Tens of Billions of Documents



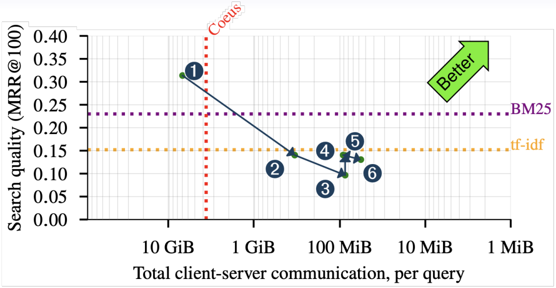
Impact of Optimizations

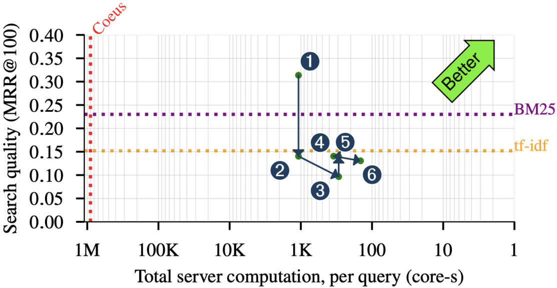
Trade-off Between Quality and Performance:

- Optimization trade-offs: embedding precision reduction, clustering, URL compression, document assignment strategy, and dimensionality reduction.
- Improvements in communication and computation at the cost of a small drop in MRR@100.

Overall Impact:

- Optimizations improve communication by two orders of magnitude and computation by one order of magnitude.





1. No optimization
2. Clustering
3. Compress URL chunks and fetch only the chuck with the top result
4. Cluster URLs in semantically similar groups
5. Assign border documents to multiple clusters
6. Reduce embedding dimension (PCA)

Discussion

Private Advertising:

- Tiptoe compatible with displaying relevant ads alongside search results.
- Clients fetch relevant textual ads using Tiptoe.
- Privacy guarantees until the client clicks on the ad.

Private Recommendations:

- Tiptoe's protocol may be useful in private recommendation engines.
- Items represented by embeddings for semantic proximity.
- Clients can privately retrieve similar items from recommendation system servers.

Discussion (Contd.)

Private Search on Encrypted Data:

- Extension to search over encrypted documents.
- Client processes corpus, encrypts embeddings and URLs, stores encrypted search data on servers.
- Search over documents while revealing no query or corpus information to the server.

Reducing Communication with Non-Colluding Services

Exact Keyword Search

Personalized Search

Related Work

Privacy-Enhanced Web Search:

- Anonymizing proxies (Tor, mix-net).
- Dummy queries or query obfuscation.
- Trusted hardware, but vulnerable to memory-access pattern leaks.

Existing Privacy-Preserving Search Engines:

- DuckDuckGo and similar engines do not track users but reveal plaintext queries.
- Coeus - Private Wikipedia search with similar security properties but higher costs.