PIR Search Using Hyperbolic Text Embeddings

Arthur Lu
UCLA

Shufan Li
UCLA

Abstract

Private Information retrieval (PIR) schemes such as SimplePIR allow users to privately fetch entries from a public database in an obvious manner. Building on top of PIR, more complex systems such as Coeus and Tiptoe has achieved oblivious document ranking and retrieval. While effective and useful, existing systems only support ranking by cosine similarity of embeddings in euclidean space. Hyperbolic embeddings have recently shown promising results in a wide range of applications such as hierarchical representation and source code search. In this work, we explore the porblem of oblivious document ranking and retrieval based on hyperbolic embeddings. One of the key challenge is the complex, highly-non-linear nature of the hyperbolic distance function, which are to compute naively using existing schemes. To address this problem, we derived an equivalent, simpler surrogate metric which always yield the same ranking as the hyperbolic distance metric, as well as a scheme to compute this metric in an oblivious manner. Experiment results show that the proposed scheme is correct and achieves better performance thanks to multithreaded optimizations and usage of specialized hardwares such as GPUs.

1 Introduction

The problem of confidential document retrieval given a user query is an active research field. Existing implementations such as Coeus [1] and TipToe [3] generally conquer this problem by dividing them into two steps. In the first step, the similarity of the user query and each document is computed in a private and secure manner. These similarities scores are then sent back to the client. In the second step, the client uses a PIR scheme to privately retrieve the actual document. Existing works typically employs some Euclidean embeddings with inner product as the similarity metric. Hence, the first step can be easily achieved using oblivious matrix-vector product multiplication.

Recent works in representation learning such as [8] suggest that hyperbolic embeddings may be a better alternative over certain domains such as code, as they are able to better represent the hierarchies of the dataset. In this paper, we present an oblivious document ranking and retrieval system using hyperbolic embeddings. The key contribution of our work is a novel computation scheme for the hyperbolic distance metric in the first of the two steps.

Concretely, we first derive a linear metric that gives the same ranking to the non-linear hyperbolic distance function. The we adopt the LWE scheme used in SimplePIR [4] to compute such metric in a secure and private manner. Lastly, we adopt several off-the shelf optimization schemes such as GPU-acceleration and multi-threading, achieving a stronger performance than previous works.

1.1 Threat Model & Privacy Guarantee

We consider a public or private database search system involving multiple client users and a single server containing some database of documents. Clients send queries to the server to search over the database and then retrieve the documents. We adopt the threat model from existing implementations where a strong adversary may arbitrarily compromise the server or the network as in Coeus. Such an adversary may have access to the database along with detailed information about the computation being performed by the server on queries and databases. We maintain the same security guarantee of *query privacy* where an adversary learns nothing about a query.

2 Related Work

Hyperbolic Embeddings have gained popularity as an alternative to Euclidean ones. [6] showed that embeddings based on a hyperbolic distance metric better represent the natural hierarchical structure of texts. [8] further highlighted its advantage in code searching applications and achieved considerable improvements in code retrieval tasks. [2] demonstrated that hyperbolic embeddings can be useful in recommendation systems. [5] further extends hyperbolic embeddings to computer vision and shows that it can represent rich visual semantics.

Given its well-recognized advantages, supporting searches based on hyperbolic embeddings can greatly improve the usefulness of private information retrieval (PIR) systems.

2.1 Secure Multiparty Computation

To achieve *query privacy*, secure multiparty computation can be utilized to compute oblivious document rankings based on a query string cipher. New methods, such as SECFLOAT, ensure accuracy in floating point math when performing 2-party computation, [7]. SECFLOAT supports correct floating-point primitive operations addition, multiplication, division, and comparison. Additionally, it supports precise trigonometric functions, exponentiation, and logarithms. While these works allow precise floating point computation of non-linear functions, they can be expensive and hard to optimize. Hence, we use fixed-point approximation in our work.

2.2 Secured Search

Many relevant works aim to provide secure document search and retrieval. Coeus [1] ranks a document's relevance to a user query using the term frequency-inverse document frequency (TF-IDF) matrix. Its protocol consists of three rounds. In the first round, a user sends an encrypted query vector and securely computes the relevant scores of documents via matrix-vector multiplication. In the second round, a user fetches the metadata of top results. In the final round, the user securely fetches a single document of interest. TipToe [3] extends the idea by replacing the TF-IDF matrix with semantic embeddings created by machine learning models, allowing fuzzy searches based on semantic similarity. It uses homomorphic encryption to compute the inner products of a query vector and document vectors and then sends the encrypted inner product scores to the client. Unlike Coeus, TipToe employs a clustering algorithm and only sends scores of the best-match cluster, achieving a $O(\sqrt{N})$ complexity.

3 Background

3.1 LWE-based encryption

Learning-with-Errors(LWE) is a common scheme used in linearly-homomorphic encryption. Concretely, the Regev-style LWE encryption is parameterized by encrypting the dimension of secret $n \in \mathbb{N}$, the number of samples $m \in \mathbb{N}$, and integer p, q where $q \gg p$. It maps plain text in $\mathbb{Z}_p$ to cipher text in $\mathbb{Z}_q$.

The private key consists of a random secrete $s \in \mathbb{Z}_q^n$. The public key is the vector $(As+e) \mod q$ where the matrix $A \in \mathbb{Z}_q^{m\times n}$ is a random matrix and $e \in \mathbb{Z}^m$ is an error term sampled from some prior noise distribution $\mathcal{X}$.

Given a plain-text vector $v \in \mathbb{Z}_p^m$, the cipher-text is computed as $C = (C_1, C_2) = (A, As + e + \lfloor \frac{q}{p} \rfloor v) \mod q$. We can decrypt the ciphertext by finding $v' = \lfloor \frac{(C_2 - C_1 s) \mod q}{\lfloor \frac{q}{p} \rfloor} \rceil \mod p$.

Given an arbitrary linear transform T, we can apply it to a ciphertext through $TC = (TC_1, TC_2)$. We can also add a plain text z to a cipher-test through $C + z = (C_1, C_2 + \lfloor \frac{q}{p} \rfloor z)$.

3.2 SimplePIR

SimplePIR [4] showed that we can reuse A without compromising security. Hence, the client only needs to send C_2, which is significantly smaller.

Specifically, suppose we have L document vectors $v_1 ... v_l$ and we want to compute the inner product $q^T v_i$ of some vector q and all document vectors ($i = 1, 2 ... L$) in an oblivious manner, the client generates a secret s, but use a pre-downloaded hint matrix A to derive the public key $As + e$. It encrypts the vector q to cipher-text $C_q = (C_{q1}, C_{q2})$. Since $C_{q1} = A$, there is no need for the client to send this matrix. The client simply sends C_{q2} which is just one vector of dimension m.

Upon receiving C_{q2}, the server computes the cipher text corresponding to the results $C_{ans} = (V \cdot A, V \cdot C_{q2})$, where $V = (v_1, .. v_L)^T \in \mathbb{Z}_q^{L\times m}$. Observe that $V \cdot A \in \mathbb{Z}_q^{L\times n}$ is independent of the query, and hence can also be pre-downloaded. The server only needs to send back $V \cdot C_{q2} \in \mathbb{Z}_q^{L\times 1}$, which is significantly smaller.

3.3 Hyperbolic Embedding

We use Poincaré disk model for hyperbolic spaces, which embed a hyperbolic space $\mathbb{H}^n$ into a unit disk $D^n = \{x \in \mathbb{R}^n; \|x\| \leq 1\}$ in $\mathbb{R}^n$. Given embeddings of two points $u, v \in D^n$, the hyperbolic distance can be computed as

$$d(u,v) = \operatorname{arccosh}(1 + 2\frac{\|u - v\|^2}{(1 - \|u\|^2)(1 - \|v\|^2)}) \tag{1}$$

4 Method

4.1 Simplifying Hyperbolic Distance

Let $\delta(u,v) = \frac{\|u-v\|^2}{(1-\|u\|^2)(1-\|v\|^2)}$, then $d(u,v) = \operatorname{arccosh}(1 + 2\delta(u,v))$, which is a monotonically increasing function with respect to δ. Hence, for the purpose of search and ranking, we can simply rank $\delta(u,v)$ and avoid computing the non-linear arccosh function.

Given a query vector q, and L document vectors $v_1, ... v_L$. We note the following relation holds:

$$\begin{aligned}\text{argmin}_i \delta(q, v_i) &= \text{argmin}_i \frac{\|q - v_i\|^2}{(1 - \|q\|^2)(1 - \|v_i\|^2)} &(2)\\ &= \text{argmin}_i \frac{\|q - v_i\|^2}{(1 - \|v_i\|^2)} &(3)\\ &= \text{argmin}_i \frac{\|q\|^2 + \|v_i\|^2 - 2q^T v_i}{(1 - \|v_i\|^2)} &(4)\\ &= \text{argmin}_i (\gamma_i \|q\|^2 + \beta_i + q^T \xi_i) &(5)\end{aligned}$$

, where $\gamma_i = \frac{1}{(1-\|v_i\|^2)}$, $\beta_i = \frac{\|v_i\|^2}{(1-\|v_i\|^2)}$, and $\xi_i = \frac{-2v_i}{(1-\|v_i\|^2)}$ are independent of the query vector q and can be easily precomputed.

Now suppose the query vector q and document vector v_i is of dimension and $q = (q_1, q_2, ..q_d)$, $\xi_i = (\xi_{i1}, \xi_{i2}, ..\xi_{id})$. We can construct a vector $q' = (q_1, ..q_d, \|q\|^2)$. Then we can compute Eq. 5 via a linear transformation $q' \to t_i^T q' + \beta_i \in \mathbb{Z}_p^{d+1}$ where $t_i = (\xi_1, ..\xi_d, \gamma_i) \in \mathbb{Z}_p^{d+1}$. To compute scores for all documents, we can simply compute the matrix $T = (t_1, ..t_L)^T \in \mathbb{Z}_p^{L\times(d+1)}$ and a offset vector $\beta = (\beta_1, ..\beta_L) \in \mathbb{Z}_p^L$, and simple compute $Tq' + \beta$.

4.2 Secure Computation

We have reduced the problem to securely compute $Tq' + \beta$, where T, β can be precomputed on the server side and are independent of the queries. This can be achieved by modifying the protocols of SimplePIR described in Sec. 3.2.

Formally, the server precomputes the hint $(T \cdot A, A)$ and sends it to the client. To perform a query, the client generates a secrete s, builds the augmented query vector q' based on the raw query q, and encrypts q' to obtain $C = (C_1, C_2)$ where $C_1 = A$. The client only sends $C_2 = (As + e + \lfloor \frac{q}{p} \rfloor v) \mod q$ to the server. The server obtains the ciphertext of the answer $C_{ans} = (V \cdot T, T \cdot C_2 + \beta)$, where $V \cdot T$ was precomputed and already included in the hint. Hence, the server only sends back $T \cdot C_2 + \beta$. The client finally decrypts C_{ans} by combining newly received $T \cdot C_2 + \beta$ and the precomputed $V \cdot T$ in the hint.

5 Fixed-Point Approximation

Since $q', \xi_i, \gamma_i, \beta_i$ are floating point numbers, we quantize them by a constant factor of $K = 256$. In particular, we scale q', ξ, γ by a factor of 16, and β_i by 256. This will scale the final score by 256. Since scaling the score by a constant does not affect the ranking result, we can directly rank the fixed point score without additional changes to the system.

5.1 Implementations

We implement the LWE protocol by extending TipToe and implementing it in Python with a CPU and GPU implementation. In the TipToe implementation, we leverage the existing SimplePIR code and matrix operations to perform the LWE protocol replacing the TipToe search protocol. In Python, we use numpy and pytorch to implement the protocol for both CPU and GPU execution.

6 Evaluation & Results

We evaluate our implementation based on CPU time and network overhead compared to baseline TipToe per query. We divide the CPU time between preprocessing time which is run only once per database, and answering time which is the time required to compute answers to queries. We also measure the user's network upload and download bandwidth requirements. Finally, we measure the hint size, which is the one-time download for each client. The evaluation was run on a 8 Thread Zen 1 machine with 8 GB of memory. The evaluation was performed on a fake corpus of 1k documents using a 768 embedding size and averaged over 1k queries. Preprocessing time and hint size are reported over all queries. Answering time, upload, and download are reported as average per query.

Implementation	Preproces	Answer / Q
TipToe Baseline	2.02 s	346 μs
TipToe Extension	2.04 s	1004 μs
Python CPU	3.00 s	604 μs
Python GPU	3.16 s	26 μs

Table 1: Preprocess Time, Answer Time

Implementation	Hint Size	Up / Q	Down / Q
TipToe Baseline	17.6 MB	16.2 MB	0.13 MB
TipToe Extension	17.6 MB	16.2 MB	0.13 MB
Python CPU	13.8 MB	5.86 KB	7.63 KB
Python GPU	13.8 MB	5.86 KB	7.63 KB

Table 2: Hint Size, Client Upload, Client Download

The TipToe extension was able to maintain the existing communication overhead, but was worse in average answering time per query. This was expected, since the server computation involved several more matrix operations than before. The client download bandwidth increase is explained by the additional return values from the server.

We also evaluate our python implementation's scaling to different database sizes. We run the Python CPU and GPU implementations on databases with 1k, 2k, 4k, and 10k documents. We keep the same embedding size of 768. For this evaluation, answering time was measured over the total 1k query response and not averaged. We plot the results in 1.

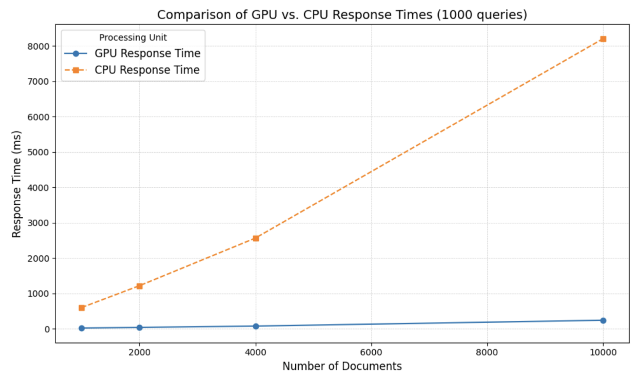

Figure 1: Python CPU vs Python GPU Answering Time

The Python implementation was able to improve on the communication overhead compared to the TipToe baseline. TipToe implemented several optimizations on answering speed by adding additional communication such as the query tokens, which the Python implementation did not. However, this resulted in higher latency per query response since more computation was required on the server. Additionally, the use of GPU power allowed for parallelization of the protocol which decreased the answer delay significantly compared to the python CPU implementation and baseline.

We also show the protocol's correctness in our evaluation against a non-private fixed point reference implementation. For each of the 1000 test queries, we compare the computed distance scores to the reference distances calculated in the plain. We plot the reference scores against the decrypted scores from the output of our protocol in 2. Note that the deviation between reference scores and computed scores is negligible, demonstrating our protocol's correctness.

We also show the error in the fixed point approximation of floating point operations. We plot the fixed point reference scores against the floating point reference scores in 3. We note that as the floating point value increases, the fixed point representation deviation increases. However, the fixed-point and floating-point score generally retain a linear correspondence.

7 Discussion

7.1 Security Guarantee

We introduced three key changes to the SimplePIR protocol. First, to compute the hint we use the derived transform matrix T instead of the document matrix V. However, this does not compromise privacy since this operation is in the preprocessing step and does not depend on user-query. Second, instead of sending the raw query q, the client sends the processed query, q' which has one additional dimension. However, this does not compromise privacy, as the LWE-based scheme guarantees the server learns nothing about the query sent by the

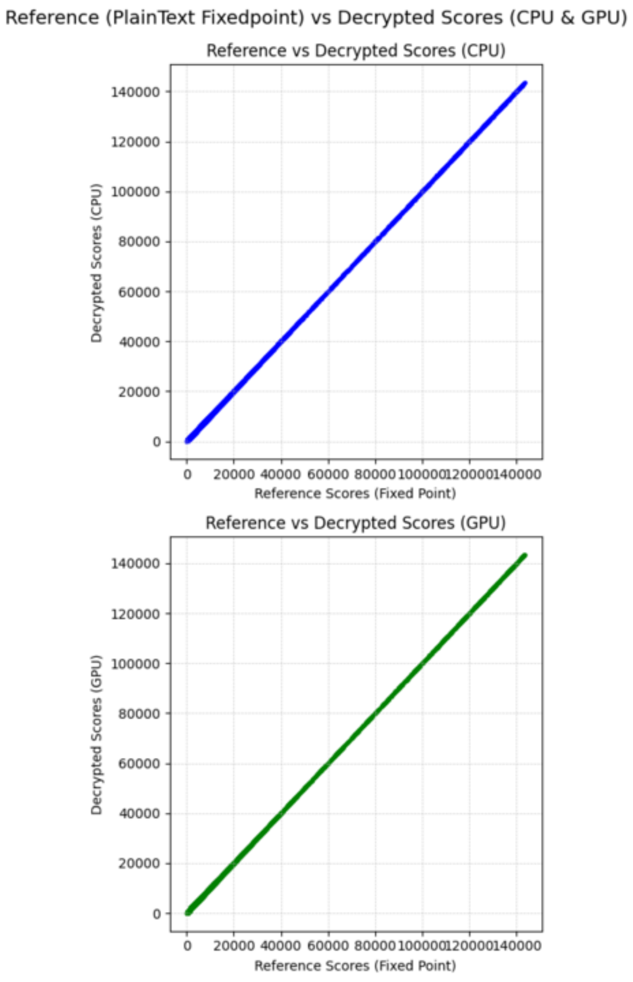

Figure 2: Reference Scores vs Protocol Scores

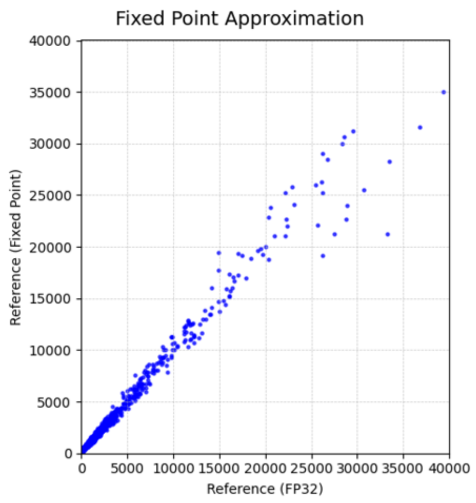

Figure 3: Fixed Point vs Floating Point

client. Since given q, we can compute q', and vice versa. Neither q nor q' will be compromised. Lastly, We add a constant β to the matrix-vector-product, instead of directly send the product back. The does not leak any additional information as the operation is query-independent.

7.2 Performance

We mostly attribute the performance gain to the off-the-shelf GPU acceleration provided by torch, and multi-threading provided by NumPy. While our method is implemented in python, it outperforms Tiptoe implemented in Go because Tiptoe calls a single-threaded, nested-loop implementation of the matrix multiplication routine.

8 Conclusion

Our project demonstrates that private information retrieval systems can efficiently implement complex search systems using custom embeddings and similarity scoring. We show that simple changes to implementation can allow existing protocols to extend support for hyperbolic embeddings and distance computation while preserving privacy. We also show that using a LWE encryption scheme can privately compute on fixed point data without compromising accuracy with bounded distance score values. We show that the impact of such additions can be implemented with only a modest impact on performance. Future work may continue to extend PIR schemes with more up-to-date search models.

External Collaborators & Previous Work

We did not work with any outside collaborators on this project. This project does not overlap with any other courses. No progress was completed on this project before the course.

References

[1] Ishtiyaque Ahmad, Laboni Sarker, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Coeus: A system for oblivious document ranking and retrieval. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 672–690, 2021.

[2] Konstantin Bauman, Alexander Tuzhilin, and Moshe Unger. Hypercars: Using hyperbolic embeddings for generating hierarchical contextual situations in context-aware recommender systems. *Information Systems Research*, 2024.

[3] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th symposium on operating systems principles*, pages 396–416, 2023.

[4] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/949, 2022.

[5] Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan Oseledets, and Victor Lempitsky. Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6418–6428, 2020.

[6] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30, 2017.

[7] Deevashwer Rathee, Anwesh Bhattacharya, Rahul Sharma, Divya Gupta, Nishanth Chandran, and Aseem Rastogi. Secfloat: Accurate floating-point meets secure 2-party computation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 576–595. IEEE, 2022.

[8] Xunzhu Tang, zhenghan Chen, Saad Ezzini, Haoye Tian, Yewei Song, Jacques Klein, and Tegawende F. Bissyande. Hyperbolic code retrieval: A novel approach for efficient code search using hyperbolic space embeddings, 2023.