QATAR UNIVERSITY

COLLEGE OF ENGINEERING

LATENT SEMANTIC INDEXING BASED DISTRIBUTED SYSTEM AND SEARCH ON ENCRYPTED DATA

BY

ABDELRAHMAN MOSSAD SHOUMAN

A Thesis Submitted to

the Faculty of the College of

Engineering

in Partial Fulfillment

of the Requirements

for the Degree of

Masters of Science in Computing

January 2018

COMMITTEE PAGE

The members of the Committee approve the Thesis of Abdelrahman Shouman defended on 13/12/2017.

Abbes Amira

Thesis/Dissertation Supervisor

Abdullatif Shikfa

Thesis/Dissertation Co-Supervisor

Professor Richard Khouri

Committee Member

Dr. Tamer ElSayed

Committee Member

Dr. Tamer M. Khattab

Approved:

Khalifa Al-Khalifa, Dean, College of Engineering

ABSTRACT

SHOUMAN, ABDELRAHMAN, MOSSAD., Masters : January : [2018],

Masters of Science in Computing

Title: Latent Semantic Indexing (LSI) Based Distributed System and Search On Encrypted Data

Supervisor of Thesis: Abbes Amira.

Latent semantic indexing (LSI) was initially introduced to overcome the issues of synonymy and polysemy of the traditional vector space model (VSM). LSI, however, has challenges of its own, mainly scalability. Despite being introduced in 1990, there are few attempts that provide an efficient solution for LSI, most of the literature is focuses on LSI's applications rather than improving the original algorithm. In this work we analyze the first framework to provide scalable implementation of LSI and report its performance on the distributed environment of RAAD.

The possibility of adopting LSI in the field of searching over encrypted data is also investigated. The importance of that field is stemmed from the need for cloud computing as an effective computing paradigm that provides an affordable access to high computational power. Encryption is usually applied to prevent unauthorized access to the data (the host is assumed to be curious), however this limits accessibility to the data given that search over encryption is yet to catch with the latest techniques adopted by the Information Retrieval (IR) community. In this work we propose a system that uses LSI for indexing and free-query text for retrieving.

The results show that the available LSI framework does scale on large datasets, however it had some limitations with respect to factors like dictionary size and memory limit. When replicating the exact settings of the baseline on RAAD, it performed relatively slower. This could be resulted by the fact that RAAD uses a distributed file system or because of network latency. The results also show that the proposed system for applying LSI on encrypted data retrieved documents in the same order as the baseline (unencrypted data).

DEDICATION

For my family, my friend Mohanned for his advices on the issues I faced, for my friend Nazar for providing his machine for part of the experiments and redbull.

ACKNOWLEDGMENTS

I would like thank Dr. Abbes Amira, my supervisor, for his continuous support throughout my thesis journey. I would like also to thank Dr. Abdullatif Shikfa, my co-supervisor for his support, non-official yet excellent, and his help especially in the security part.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF PUBLICATIONS

As part of this work we plan to submit two publications. First publication will discuss our experiment on LSI scalability. The publication will discuss the experiment of applying gensim's implementation for LSI on RAAD's distributed environment. It will also report the conducted results including insight, challenges and limitations. For this publication the JPDC[1] journal is targeted. Second publication will revolve around our proposed method to apply LSI on encrypted data. For this publication, the IET Information Security[2] journal is targeted. The publication will discuss the results of the proposed method and how does it fit in the literature and advance the existing methods. This includes the switch to LSI model instead of simple occurrence matrix and adaption of free-text queries instead of simple keyword query.

[1] https://www.journals.elsevier.com/journal-of-parallel-and-distributed-computing
[2] http://digital-library.theiet.org/content/journals/iet-ifs

Chapter 1: Introduction

1.1. Introduction of Information Retrieval (IR)

The urge need for efficient retrieval techniques emerged with the introduction of web search. Retrieving user needs requires searching across a large scale of data, which is a challenging task that led to continuous development of retrieval models. Prior to that, searching was performed in the style of Database (DB) search where providing precise information is vital to acquire effective results. For example, searching in a cars' DB requires detailed information like plate number. Similarly, searching a patients' DB necessitates patient national ID number. In the former example, the search patterns involved limit the usage of the system to people who are trained for that purpose, such as administrators, customer service advisors, paralegals, and so forth.

Web search shifted the older paradigm where the majority of users are not trained professionals with pre-knowledge of the data. It is rather a broad group of users with different backgrounds and various information needs. Queries formed by those users are unstructured and described as free text queries. The whole problem of Information Retrieval in the academic field is described as following:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)" [1].

This new paradigm deals with documents on a larger scale that requires keeping a sense of term weighting. Terms are the result of preprocessing all words in the document collection. The preprocessing might include stemming or stop words removal. During query-time, term weighting is vital to calculate a score relevance with reference to the query. The higher the score the more relevant the document is to the user's need; hence, retrieval models will list the most relevant documents at the top of the results. The way it works is that term weighting is used to form a document vector representation that "captures the relative importance of the terms in a document"[1]. Queries are represented using the same vector notion as well, where vectors similarity measures (e.g. cosine similarity) can be leveraged to apply a search query. The set of document vectors represented in the same vector-space is called Vector Space Model (VSM) [1].

1.2. IR's Common challenges

The aforementioned model is keyword dependent. Documents that share common query terms are more likely to be retrieved as relevant in comparison to documents that do not. The main deficiency of this approach is the lexical mismatch between the users (searchers) and the authors of retrieved data, which results in two main issues: synonymy and polysemy [2]. First, multiple words can have the same meaning (e.g. Chair and Seat share the same meaning) and there is no guarantee that the user will search for one term over the other. This problem of Synonymy can lead to less relevant documents being retrieved, hence, decreasing the system's recall. Second, a single word may have multiple meanings (e.g. Jaguar can refer to the car brand or the animal) and there is not straightforward way for the model to detect the main intention of the user. This problem of Polysemy implies that some non-relevant documents will be retrieved, thus decreases the precision of the system. Both Recall and Precision are significant metrics to measure the relevancy of the system which is one of the core concepts for information retrieval.

Two solutions were suggested to overcome the issues of synonymy and polysemy according to [3]. First approach is Stemming, which is a process to convert words to their morphological root so ‘retrieval’ and ‘retrieving’ would be converted to ‘retrieve’. However, this process does not work to match nouns like chair and seat. Second is Controlled Vocabulary; this solution suggests that document authors and searchers use a predefined set of terms. However this issue was described by [4] as both expensive and inefficient.

Three factors were proposed by [4] to justify the failure to address the issues of synonymy and polysemy. First, indexes lack enough terms; this can occur because of the limited terms used in the documents collection itself or because of the preprocessing phase that omits many terms. Second, there is no practical methodology for handling polysemy; one solution was controlled vocabulary, however as previously mentioned, it is both inefficient and expensive. Third, terms are not dependent on each other; meaning that two terms frequently co-occurring have the same retrieval possibility as those rarely co-occurring. Hence, queries including compound terms such as check-in are more difficult to satisfy.

1.3. Introduction to Latent Semantic Indexing (LSI)

The discussion above describes an existing gap resulted because of the loss of the meaning of query terms. While constructing a query, the user usually has a need they wish to satisfy, and if the query terms are not adequately representative of those needs (i.e. they miss the meaning or the user's intention) the systems performance necessarily drops. Latent Semantic Indexing (LSI) was initially proposed to cover that gap, it assumes the existence of a latent semantic structure covered by a wide variety of word choice that sit its goal to uncover that structure [2] [3]. Thus, for a certain query, a relevant document that does not have any query term can still be retrieved.

LSI provides a step further compared to the old VSM. To understand how LSI is an improvement to the VSM model, it is important to be familiar with how VSM works. Following are the common steps of both LSI and VSM in addition to the LSI's additional steps

A. **Pre-processing:** The preprocessing phase includes steps like stop words removal and stemming. These steps are applied on the original corpus of documents (or other searchable material) to extract the important terms form it.

B. **Term document Matrix:** Terms generated from the previous phase are then used to generate the term document matrix A. Records in the matrix are the terms, and columns are the documents. The value stored in each individual cell represents the occurrence of a certain term in a specific document.

C. **Matrix decomposition:** The output term-document matrix A is highly sparse with unnecessary noise. In this third phase a new matrix of lower dimensions and rank (k) is approximated[3] from the original term document matrix (with rank r). The low-rank approximation technique aims to find a matrix with the lowest possible discrepancy from the original term-document matrix. Rank k is manually selected and it has to be

[3] This approximation can be referred to as the LSI model or Index.

significantly lower than r. The approximation is applied using a matrix factorization technique called singular value decomposition (SVD), which represents the matrix as a product of metrics derived from its eigenvectors. The equation below shows the SVD factorization applied on A.

$$A = U\,S^{4}\,V^{T}$$

Equation 1: SVD applied on the term-document matrix A

These three matrices are, U and V^T, left and right singular vectors of A; a set of orthonormal eigenvectors and S, singular values of A.

$$U\,S\,V^{T} = [u_1 \quad u_2]\begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}\left[\begin{matrix}v_1^T\\ v_2^T\end{matrix}\right]$$

Equation 2: Left: Left Singular vectors, Middle: Singular Values, Right: Right Singular Vector

If the aforementioned rank k is selected to be 1, then A_k will be as following.

$$A_k = [u_1 \quad 0]\begin{bmatrix}\sigma_1 & 0\\ 0 & 0\end{bmatrix}\left[\begin{matrix}v_1^T\\ 0\end{matrix}\right]$$

Equation 3: A_k Calculated

[4] Σ is sometimes used instead of S

SVD is the value LSI adds to the conventional keyword dependent retrieval method using term-document matrix.

The matrices are then lowered to a rank k where the first k vectors (in case of U and V^T) or values (in case of S) are selected. Those k-ranked matrices are then multiplied to calculate an approximated matrix A_k that is used instead of A to search. This low-rank approximation matrix helps to cluster a given set of documents based on the co-occurring terms. This means that similar documents (i.e. documents with similar topic and in turn terms) are represented closer to each other in the LSI vector space. Therefore, a given query can retrieve documents that do not share terms with it and thus, generating a model that is keyword-independent.

1.4. LSI Hello World Example

The following example illustrates how LSI is able to recognize similar topics among documents that may not share common keywords.[5] Table 1 lists nine different document discussing two topics. Documents c1-c5 are about human-computer interaction (HCI) and documents m1-m4 are about graph theory.

[5] The original experiment was conducted in [4]

Table 1. List of Documents Used in the Experiment Conducted by [4]

DOCUMENT CODE	DOCUMENT TITLE
C1	Human machine interface for lab abc computer applications
C2	A survey of user opinion of computer system response time
C3	The EPS user interface management system
C4	System and human system engineering testing of EPS
C5	Relation of user perceived response time to error measurement
M1	The generation of random binary unordered trees
M2	The intersection graph of paths in trees
M3	Graph minors IV Widths of trees and well quasi ordering
M4	Graph minors A survey

Gensim library was used to generate the LSI model, more discussion about Gensim will be provided in section 4.1.1.6.

Figure *1* below is a geometric representation of the resultant LSI model in two dimensions. According to [5] the list of documents in Table 1 were deliberately designed to “produce a satisfactory solution using just two dimensions”.

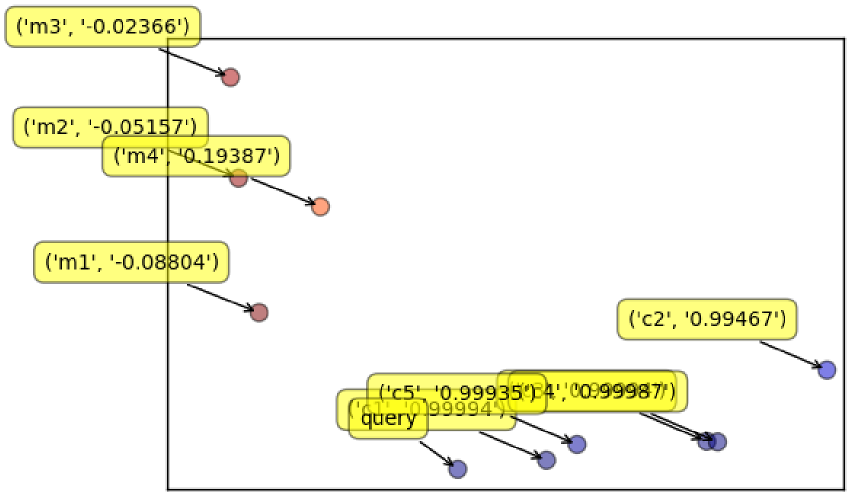


Figure 1. Two-dimensional representation of the LSI model generated from the documents in Table 1

The Space model depicted in the figure above shows how the documents are clustered in two different groups; HCI and graph theory. As aforementioned, documents labeled ‘m’ discuss graph theory and documents labeled ‘c’ discuss HCI, and each document was clustered into its respective group based on its topic.

Figure *1* depicts the cosine similarity between each document and the query “Human computer interaction”. HCI group expectedly achieved higher scores and the query 2-dimensional representation is plotted closer to the HCI group as well. It is worth mentioning that documents ‘c3’ and ‘c5’ share no term with the query, nevertheless the LSI model managed to detect their similarity with the query.

1.5. Problem Statement

It is difficult to for data owners/authors to make their data accessible for their target customers/users without enduring excessive costs. Once emerged, cloud computing provided a good cost-efficient alternative that minimizes management efforts. It can host large scale datasets while providing search functionality to customers. However, cloud services require full access over data, which might not suit its owners. The goal is to design an LSI system that can effectively search over encrypted data in a distributed setting that support scaling.

Two main challenges present themselves as obstacles to achieve this goal. Firstly, LSI suffers from a scalability issue. This is because it is not easy to parallelize SVD, the main component of LSI. As a result, LSI performance suffers over large datasets. Second, while encryption might secure data from unwanted access, it also prevents cloud services from performing any form of computation.

1.6. Research Objectives

The main objective of this thesis is to investigate the possible settings of applying LSI in a distributed environment and on Encrypted data. If successful, it can potentially help applying highly efficient search on the cloud functionality while ensuring the confidentiality of the data. Following is a list of this work’s objectives:

- To research LSI’s performance on distributed systems and Encrypted data
- To research variations of LSI and its applications
- To carry on an empirical study of LSI in a distributed system environment
- To investigate how LSI can be applied on encrypted data

1.7. Research Contributions

Following is a list of the main contributions of this work

- Conduct a literature review of the main variations of LSI. For each of those variations, their signification over the original LSI is reported and their potential to scale in a distributed environment is discussed. A summary of LSI applications is presented alongside discussion of the LSI variation and dataset(s) used.
- An experiment of LSI is carried on and HPC environment in serial and distributed mode to evaluate both performance and effectiveness of gensim’s LSI (check section 4.1.3.1 and 4.1.3.2). The evaluation is carried on using two reported measurements; CPU time and average precision (check sections 4.1.24.1.4.1 and 4.1.4.2).

- A study of how LSI can be applied on encrypted data is presented as well. A list of different experimental attempts is reported along with environment's setting (check section 4.1.44.1.3). Each attempt is then reported and compared against a devised baseline (check section 4.2.3.2).

1.8. Research Motivation (Significance to Local Community)

It is important for the work presented in this thesis to be aligned with strategies of local education and research institutes in Qatar. The section shows how this work achieves such alignment.

1.8.1. Qatar National Research Strategy (QNRS)

Qatar National Research Strategy (QNRS), is a publication of the Qatar National Research Fund (QNRF). It reflects input from Qatar research leadership, researchers and other stakeholders [6]. QNRS regularly highlights various key challenges that faces Qatar to help advance certain research areas. Research and educational institutes' effort and fund is then directed toward these focus area; thus, becoming a country-level priority.

1.8.1.1. Powerful and Distributed Computing

QNRS has published a pillar regarding Computer Sciences and Information Technology (ICT). One of the listed goals (Goal ICT.2) was to "Build a research program on distributed, data intensive and service oriented computing" [7]. QNRS also highlighted the importance of big data computing where they advised to "Explore and implement best practices in Big Data Urban development project" [8]

In part of this work, an experiment of LSI in distributed systems is carried on as mentioned in the research objectives section. The experiment runs in a distributed environment on a large dataset (check section 4.1.3.1). This fits QNRS intention to focus on big data and distributed computing. The distributed environment is provided by RAAD, a High-Performance Computing (HPC) system provided by TAMUQ[6] (check section 4.1.1.8 for more details). This also fits QNSRF to focus on data intensive computing.

1.8.1.2. Cyber Security

Cloud computing has witnessed rising importance over the years, it was strongly highlighted as a priority research area by QNRS. This shift also introduced security concerns on a nation scale according to a QNRS publication [9]. Cyber security was therefore announced as a "a National Grand Challenge"; thus, leading to a higher focus and larger fund in this area.

In part of this work, a proposal of how LSI can be applied on encrypted data is presented. The proposed solution, although not finalized, has the potential of advancing how cloud computing is used. In other words, encrypted data can be outsourced to cloud services while still maintaining the ability to search it using LSI.

1.9. Thesis Structure

Chapter 2 further discusses the problem definition, extending the problem statement. In chapter 3, a literature review of various LSI variation, their scalability capabilities and their applications are presented. Chapter 4 describes both experimental parts along with the detailed description of used datasets, applied preprocessing steps (if

[6] Texas A&M University in Qatar

any) and baselines. Chapter 5 provides the experiment result of formerly described parts while chapter 6 concludes the work and suggests future work.

Chapter 2: Problem Definition

This chapter provides an elaboration of the problem statement formerly reported. It starts with an overview of the challenges introduced by adopting the cloud paradigm. It then provides a description/definition of the problem based on those challenges. After that attempts to address those challenges are described, categorized along with this work's contributions in each category, if any. These contributions are then reflected in the research methodology detailed in Chapter 4: Research Methodology.

2.1. Background

Access to high computational power is not an issue for multi-billionaire organizations. Companies like Facebook, Google and Amazon have their own data centers where they store their data and perform the necessary business operations. On the other hand, running such large-scale centers can be a financial burden for small companies, startups and research groups.

In the latter case, cloud services can provide an affordable access to high computational power resources while mitigating the burden of management and maintenance. Success stories for both Amazon[7] and Google[8] cloud services include well known profiles (e.g. London Heathrow, Spotify, Airbnb, Air Asia, etc..) which shows how appealing cloud services are.

However, maintaining confidentiality while leveraging cloud services is

[7] https://aws.amazon.com/solutions/case-studies/all/
[8] https://cloud.google.com/customers/

challenging for several reasons. First, cloud services need to have full data access to carry on the required operations. For example, a business hosting its services on cloud servers will be sharing its customers' data (including sensitive financial information) with untrusted third party (i.e. cloud service provider). Second, some providers maintain the right to sell data or part of it for other business, which might expose user information, with or against their will. Third, cloud services usually store huge amount of data, which make them more attractive to hackers and thus relatively more prone to hacker's activities, deeming them less trustworthy [10].

The standard solution to address confidentiality concerns is to encrypt all data locally before outsourcing it to the cloud. This will prevent unauthorized access to data including both insiders (i.e. service providers) and outsiders (e.g. hacker, data purchasing) [10]. However, this would also stop data owners and authorized users from performing operations like searching using cloud resources[9].

One possible method to guarantee access on encrypted data is to first download data, decrypt it locally, and then perform the search. Yet, this method is inefficient and deprives owners from the computational capabilities of the cloud. Not to mention the additional cost of data transfer from the storage service. Another method is to perform decryption on the cloud, run the query and return results. However, this allows the server to learn the plain-text information stored; thus, making the encryption less effective [10].

[9] The search functionality is the operation of interest in this work.

2.2. Searchable Encryption (SE)

Searchable Encryption (SE), refers to the server ability to search over encrypted data without learning information about the plaintext data [10]. Several solutions have been proposed in the literature to apply SE.

An IR system (even one that does SE) consists of two main components; indexing and retrieval. First, a collection of documents is indexed to generate a matrix that represents the collection (i.e. an index). In such a matrix, a column is a representation of a document and referred to as a vector. Different matrices can be generated based on the application. This includes, term-document matrix, an occurrence matrix, tf-idf matrix (see section 1.3) and co-occurrence matrix (LSI). Second, a user formulates a query. The query, which is treated as pseudo-document, is then transformed to a vector where the transformation is dependent on the matrix used. Similarity calculations are then applied to determine the most relevant documents.

This section briefly discusses implemented techniques in both components according to the literature and how this work fits in.

2.2.1. Indexing

Most of discussions presented in the literature focus mainly on how to encrypt the matrix not which matrix to use. The selection of index in the literature is usually occurrence matrix.

The focus of this work is LSI; thus, the matrix of selection is the co-occurrence/approximation matrix A, which is resulted from applying SVD on the term-document matrix. This work aims at investigating how A can be hidden from the cloud server while maintaining both effectiveness and efficiency. It assumes the use of multiple cloud servers

2.2.2. Retrieval

In IR systems users are expected to form free-text queries as described in section 1.1, however, SE systems are more restricted. Some systems accepts only a single keyword query [11] while others extend it with the help of operators. The combination of a keyword and an operator like AND, OR, NOT is called Boolean search. Conjunctive search is one type of Boolean search, it can be described as **AND of ORs** (e.g. $[A \vee B] \wedge [C \vee D]$). Another type is disjunctive search which is an **OR of ANDs** (e.g. $[A \wedge B] \vee [C \wedge D]$). Negation is also a Boolean search represented as $\dashv A$. Authors of [12] present their work as the first to support all three conjunctive, disjunctive and negative search. According to [12], previous proposed SE systems either focus only on conjunctive search [13][14] or disjunctive search [15]. In this work, query is assumed to be free text like search engines.

2.3. LSI Model Generation

As mentioned earlier, we plan to use the LSI model instead of the typical occurrence matrix. However, if we plan to use real data, it is important to ensure LSI can scale. LSI is well-known for its scalability issues (see section 3.3) which explain the attempts to propose variations of the original method (see section 3.1). Most of those proposals are bounded to theory or small-scale proof-of-concept experiments. However, the work of [16] introduces a “sandbox environment” or a framework that provides scalable implementation to various topic-modeling algorithms as discussed in 3.1.4.

The proposed framework report results of LSI that are highly efficient on large-scale data. In this work, we analyze the performance of that framework under different circumstances in a distributed environment of our selection. Findings and limitations (if any) are then reported.

2.4. Problem Description

Data owners (i.e. research groups, companies, etc.) need to securely store documents[10] on cloud which is more feasible than other options; thus, they need to encrypt their documents. However, this might limit the capability to search over data. The goal is to effectively let the server search over encrypted large-scale data on behalf of the client (i.e. data owner or authorized user) without learning the plaintext data [10]. According to [10] and [12], the service providers are assumed to be semi-honest; it would perform operations but still want access.

A proper solution for this problem should implement a good SE schema; a schema that only lets the server learn access and search pattern as described in [17]. Access pattern refers to which files have been retrieved, and search pattern refers whether two searches were performed by the same keyword. In addition, it should overcome the traditional challenges of LSI (described in section 3.3) and scale to large datasets.

This is reflected in the research methodology as two parts; Part 1, investigates LSI ability to scale noting the challenges mentioned in section 3.3 and Part 2, investigate the possibility of leveraging LSI in a SE system.

[10] Any indexable entity is valid (e.g. music, pictures can be represented in a document containing meta information about them)

2.5. Solution Overview

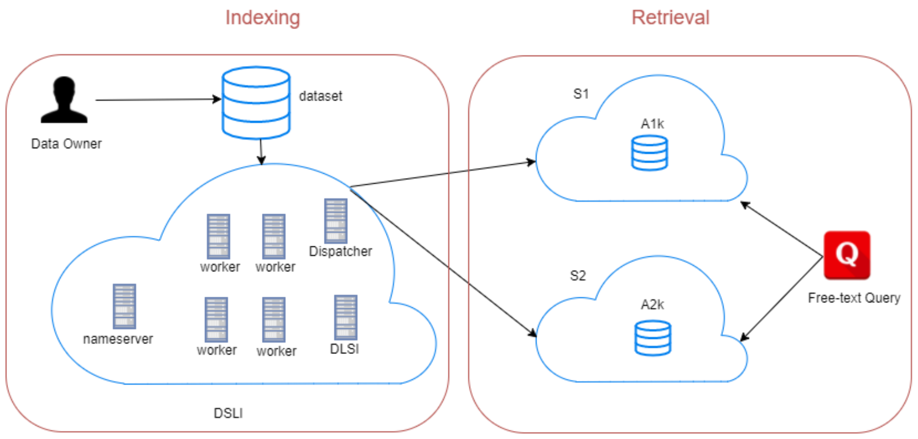


Figure 2: Solution Overview

The figure above shows a vision of a proper solution for the problem discussed in the previous section. The indexing part shows the dataset indexed using a distributed LSI environment discussed in detail in section 4.1.3. The second part shows the approximated matrix of LSI, A_k, splitted into two cloud servers and retrieved using a free-text query as discussed in section 4.2.3.

Chapter 3: Literature Review

In this chapter, the variations of LSI are discussed, those variations are attempts to overcome LSI shortcomings which is explained in this chapter as well. Furthermore, these variations support for parallelization is discussed if any. In addition, an overview of LSI applications is provided. It is worth noting that most of the literature focused on LSI applications rather than improving the algorithm itself.

3.1. LSI Variations

LSI scalability issues are caused mainly by its orthonormal constrains according to [18]. LSI's main component, singular value decomposition (SVD) factorizes a term-document matrix A to three matrices (as shown in Equation 1).

A constraint in this context refers to a preceding priority on one of SVD factorized matrices. In the case of LSI, it has an orthonormal constrain on both U and V^T; meaning that both matrices must be orthonormal. These constraints make it difficult to run SVD (and in turn LSI) in parallel while maintaining the orthonormal property. This challenge initiated the research community to introduce various alteration to the original method of LSI to overcome its scalability issues.

3.1.1. Probabilistic Latent Semantic Indexing (PLSI)

The model of PLS, as first introduced by [19], assumes that a document is a mixture of latent classes (i.e. topics). Each word w in a document d is associated with a latent class z. It represents the joint probability of w and d as following:

$$P(d, w) = P(d|z)\, P(z)\, P(w|z)$$

Equation 4

Equation 4 can be represented as a matrix factorization as shown in Equation 1 where U is $P(d|z)$, Σ is $P(z)$ and $V^T$is $P(w|z)$.

In contrary to SVD, U and V in PLSI have no orthonormal constraint which means implementing PLSI in a distributed framework is possible. Various implementations of PLSI are available for research purposes on GitHub like [20] and [21] however they not production-ready to be utilized on an HPC platform.

3.1.2. Regularized Latent Semantic Indexing (RLSI)

The main motivation behind RLSI is to address the scalability issues in LSI and PLSI according to [18]. The orthogonal property and the probability distribution of LSI and PLSI respectively resemble a constraint on both algorithms in the context of implementation on parallel and/or distributed environment.

Regularization is a technique used to solve the overfitting problem that tries to minimize a loss function. In [18] two regularization terms were introduced V and U that corresponds to the U and V of SVD. V is a topic-document matrix where v_{Kn} represent k^{th} topic weight in the n^{th} document. U is a term-topic matrix where u_{mK} represent m^{th} term weight in the k^{th} topic. The loss function is an approximation of document d_n. The optimization problem RLSI amounts to "formalizes topic modeling as a minimization of a quadratic loss function with a regularization (either $l_1$or l_2 norm)" [18].

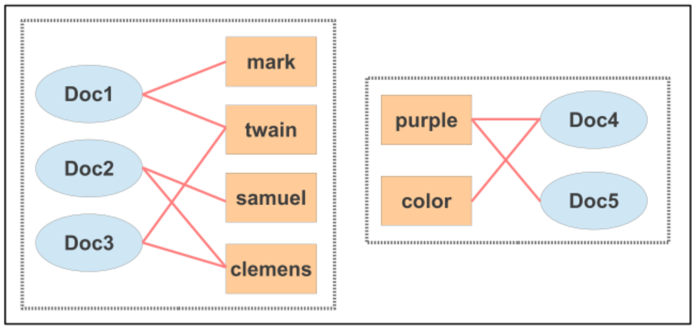
Experiments show that in terms of interpretable topics and relevance ranking, RLSI is better than or comparable with LSI and PLSI. RLSI can also scale up to 1.6 M document and 7 M terms. This technique was further enhanced by the same team in [22] where a new regularized technique named L1/2 regularization was introduced. The authors claim that said method was used to avoid both constraints of LSI and PLSI.

An existing solution of RLSI is developed by one of the authors along with a development team [23], however similar to PLSI it is more suitable for research purposes.

3.1.3. Similarity-based Matrix Completion Algorithm for LSI

This algorithm provides an alternative for SVD through mimicking its capability of solving the synonymy and the polysemy problem. The algorithm proposed in [24] leverages Bipartite graphs (i.e. bi-graphs) to understand how the relationship (i.e. weight) between documents and terms evolve. Both documents and terms are represented as vertices and term-document weight as the weight of an edge.

Figure *3* shows an example of Bipartite graph that depicts representation of a term-document matrix and its LSI approximation. New connections (edges) are constructed between terms and documents based on the terms co-occurrence.



(a)

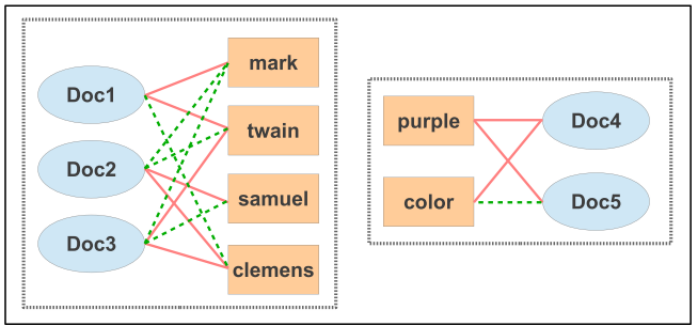


Figure 3. Bipartite graph representation for (a) the original matrix and (b) the approximate matrix. The dashed lines in (b) represent new connections due to lower rank approximation to the original matrix.

The proposed solution compares both graphs to understand the changes and then build a converging algorithm to mimic those changes. It is unclear if the work of [24] can be implemented on a distributed environment as there is no mention of parallelization's support in the paper. In addition, no implementation of the algorithm has been made publicly available.

3.1.4. Subspace Tracking for LSI (Distributed LSI)

3.1.4.1. Overview

In [16], the authors express the need for a topic modeling framework based on the analysis of existing implementations as they are not scalable, nor are they easy to use. The “lack of a sandbox environment” which can be used on real data and meet the “the high computational demand” of these topical methods has partly contributed to crippling the public from adapting topical methods. Factors like “the inherit mathematical complexity” have also extended the gap between research and practice [16].

This topic modeling system can be visualized in three phases shown in the graph below. First, a term-document matrix is generated using the provided dataset. Second, a topic modeling algorithm (e.g. LSI, Latent Dirichlet Allocation (LDA), etc.) is applied where the output model, or documents representation, is then used to calculate similarity against a user issued query. The framework proposed in [16] aims to scalably calculate these document representations distinguishing itself from the work of [25] which focuses on scalably computing pair-wise document similarities from existing models.

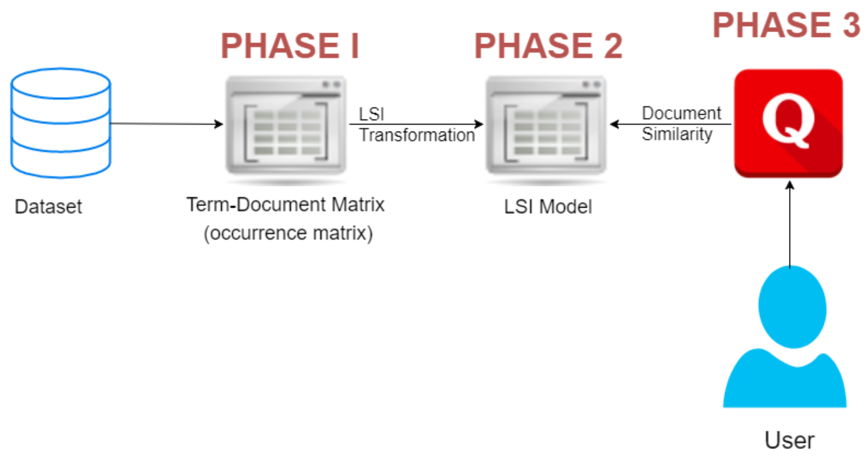


Figure 4: Topic Modeling Phases

The proposed solution also differs from the existing toolkits like NLTK, Weka, Orange and others which are "mature", yet suffer from one or more shortcomings [16]. First limitation is a lack of topic modeling as these packages usually provide supervised learning functionality (e.g. classification) in contrast to the unsupervised topic inference. Second, lack of scalability, where the package requires loading the whole corpus in memory at one point. Third, different domain focus, where the package is initially created for domains like physics, neuroscience, etc. Fourth, grand unified frameworks; frameworks designed to cover a broad range of algorithms and use cases. While those frameworks are more appealing to users, they can affect the quality of the final product.

3.1.4.2. Proposed Framework

The proposed framework has two main interfaces, corpus and transformation. First, corpus is an interface designed to handle a sequence of documents represented in occurrence vectors (i.e. occurrence matrix). The main goal of the interface is to ensure that "at no point is there a need for the whole corpus to be stored in memory" [16]. It also supports loading and storing matrices to a disk so there is no need for repetitive generation of the corpus. Second, transformation refers the process of translating documents from one vector space to another (e.g. occurrence matrix to LSI matrix).

The authors of [16] listed four system design choices for their proposed framework. First is corpus Size independence; meaning that the framework should be able to handle corpora that is larger than the machine's RAM. Second is intuitive API; which implies a use of NLP-related terms and minimal need of method names and interfaces to use the framework. Third is easy deployment; which means that the framework should not need root access and should be OS independent. Fourth is for the framework to cover popular algorithms (e.g. TF-IDF, LSA, LDA, etc.) and provide a novel and scalable implementation of those algorithms.

Python was chosen as the programing language for this framework. It helps achieve the set choices as it is OS independent, easy to implement and has a "straight-forward and compact" syntax [16]. Python also has a fast-numerical library, numpy (see section 4.1.1.2) which is widely used in this framework.

3.1.4.3. SVD Challenges

As formerly mentioned, algorithms' implementation need be scalable and memory independent. In this work only LSI is of interest, whereas the rest of this section focuses on providing a brief overview of LSI's scalable implementation used in [16].

LSI main performance challenge is because of SVD. Due to its computational intensiveness, it is expensive to calculate SVD on large datasets [26]. Packages like PROPACK and SVDPACK provided highly optimized implementation of SVD according to [16], however, they require the whole corpus to be loaded in memory. The authors of [16] were looking for alternative implementations of SVD. The idea is to allow incremental SVD, meaning that SVD can be calculated from a document stream rather than a batch of documents. Some works like [27] and [28] have proposed algorithms to incremental SVD. The work of [27] was found to be too slow and hard to tune while [28] is relatively faster and requires no tuning according to [16].

3.1.4.4. Targeted SVD

Since there is no publicly available implementation of the work of [28], the authors of [16] provided a pure python implementation of their SVD algorithm which is explained in [29]. Five characteristics were listed in [29] where its authors claim that their work has exclusively met all of them. First, being Distributable, which is a reference to the ability to run the algorithm in parallel on server autonomous nodes with no necessity for further modifications. Second, Incremental Updates, as previously mentioned it refers to the ability to update SVD once new data arrives (i.e.

support for document streams). Third, leveraging Matrix Structure; meaning that the algorithm should make use of sparse matrices (or sparse nature in occurrence matrices). Therefore, it is preferred that the algorithm are expressed in terms of Basic Linear Algebra Subprograms (BLAS) since they will adapt more easily to different type of inputs [29]. Fourth, Subspace Tracking; this means the in case of a document stream, any new observation should be immediately processed and discarded. Fifth, Available Implementations, as such, the work of [29] is open sourced and available as part of the framework developed in [16].

3.1.4.5. Distributed LSI

The distribution characteristic is achieved through column partitioning. Matrix A is divided into sub-matrices called jobs, $A^{m\,x\,n} = [A^{m\,x\,c_1},\ A^{m\,x\,c_2}, \dots ,\ A^{m\,x\,c_j}$. Every sub-matrix (i.e. group of column/documents) amounts to a processing chunk c_j. The bigger the chunk, the faster the processing is. However, bigger chunks require larger memory, thus, chunk size is dependent on available resources.

Jobs are then distributed on the available cluster nodes (i.e. workers or nodes running the worker script). The algorithm does not require a specific order to distribute those jobs. It also does not require nodes to process the same number of jobs and it does not process them at the same speed. Computations on the nodes are completely asynchronous, where each node computes the decomposition for each job it receives. Later, decompositions are accumulated and merged into a single decomposition. In short, two decompositions are performed; first, base decomposition, which occurs in memory and computed for each job, and second, merge decomposition which occurs on one node to merge all other decompositions.

3.1.4.6. This work

This is the largest section in the literature review. The reason is that, in this work, the algorithm described in [29] and introduced in the framework of [16] is the one leveraged to show if LSI can be scaled up on real data. This the only work to provide a "sandbox environment" as previously mentioned. Additionally, it is open sourced, so it can be altered to the needs of this thesis. A discussion of LSI performance on real large datasets will be later discussed and compared to the results the framework developer reported in [30].

3.1.5. Comparison of LSI Variations

Table 2: Comparison of LSI variations

	CONSTRAINTS ON *U*	**CONSTRAIN ON *V***	**SUPPORT FOR PARALLELIZATION**	**AVAILABLE IMPLEMENTATION**	**PRODUCTION-READY (FOR DISTRIBUTED ENVIRONMENT)**	**OPEN-SOURCE**
LSI [4]	Orthonormal Constraint	Orthogonal Constraint	No	Yes	No	Yes
PLSI [19]	Probability Distribution	Probability Distribution	Yes	Yes	No	Yes
RLSI [18], [22]	-	-	Yes	Yes	No	Yes
MATRIX COMPLETION [24]	-	-	N.A.	No	No	No
SUBSPACE TRACKING [29]	-	-	Yes	Yes	Yes	Yes

3.2. Applications

3.2.1. Collaborative Filtering

Collaborative filtering is one of the fields where LSI was efficiently applied as reported in [31]. The rise in the number of web services gave birth to many portals that aim at assisting users to find web services of interest. The search functionality in these portals is based on syntactical matching of both query terms and web services' descriptions. However, this text-based approach encountered semantic-based synonymy and polysemy. It is possible to address the syntactical limitations using semantic annotation markup languages. However, this approach is time consuming and is done by domain human experts. The author of [31] proposes a recommender system that uses user's id and WS operations instead of queries and descriptors. The shift of focus from text to user's behavior resulted in exploiting meaningful implicit knowledge as reported in [31].

The proposed solution consists of three main steps. First, processing users' data, where users are treated as documents and operation are treated as terms in the term-document matrix model. Second, an m * n TF-IDF matrix A is constructed where n refers to the number of users and m to the number of operations. In this matrix a cell $A_{i,j}$ is the weight of the operation O_i for the user U_j. An instance of an operation could be $getWeather$, $getlocation$, $getNews$, $getWeeatherByZipCode$, etc. LSI is then applied on the TF-IDF matrix to generate an approximation of it in a k-dimensional space. Third, given an operation O_i for a user U_j a recommendation is generated by first finding similar users then finding similar operations within the collection of those users' operation. The third step was applied using both the TF-IDF matrix (VSM-based approach) and the LSI matrix (LSI-based) approach. According to [31], LSI-based approach achieves better predictions than the VSM-based with small k. Moreover, the best prediction using LSI-based approach is always better than the best prediction using VSM-based approach.

Another example of applying LSI to implement collaborative filtering is the article recommender for Digg articles proposed in [32]. Digg is a social platform for sharing articles including blogs and news articles. Users can express interests in articles through digging them while burying articles they did not like. A score for each article is calculated based on diggings subtracted by buryings. Scores and numbers of diggings are then used to determine which articles appear first to the user. Recommending articles based on user's interest is a challenging task since Digg does not keep track of articles' topics. The proposed system in [32], DIGTOBI, leverages PLSI generative model with respect to latent topics in order to construct topics based on articles' description. If a user dug an article, then topics relevant to it would be preferred over others, even if the article does not have a high score. Experiments have shown that DIGTOPI managed to outperform state of art techniques in the literature in all used evaluation metrics.

3.2.2. Clustering

Another fields where LSI was applied is Clustering. In [33] LSI was leveraged in the context of web clustering as using search engines can be time and effort consuming in case of ambiguous information. For example, when searching using the query "jaguar" the results would be mixed of "big cat" and "car brand", however if the user's main intention was either “comic” or “music” the query would need to be rephrased. LSI was adapted to overcome the synonymy and polysemy problems encountered by the SRC clustering methods that uses bag-of-words method. In the proposed method, LSI is aggregated with Agglomerative Hierarchical Clustering (AHC). It is a bottom-up approach where cluster pairs are merged as one moves up the hierarchy. AHC was preferred over flat clustering for three main reasons. It does not require the number of clusters as an input, it returns a more informative hierarchy and the user does not need to read all the topics in order to find the desired one.

LSI is one of three steps of the proposed method. First, a collection of search results with snippets is retrieved and filtered using preprocessing techniques (e.g. Tokenization, stemming, etc.). Then LSI is applied on the retrieved data with variation of weighting schemes and k parameter. A variation of SVD was used as well where the approximated matrix is only the product of $\Sigma * V^T$ instead of $U * \Sigma * V^T$. To evaluate the proposed method, it was applied on two datasets while four different performance metrics were used. The results show how LSI managed to boost the performance compared to other clustering techniques.

3.2.3. Geographical Taxonomy

LSI was used as well to enhance retrieval in spatial domains. In [34] LSI helped to build a geographical taxonomy of adjacency for a given country. According to [34], the need for such taxonomy raises from the ambiguous nature of query while knowing that one query of five has a geographical context. This means that a geographical taxonomy can be used to reformulate the query to be less ambiguous. The proposed method supports both absolute and relative special entities (ASE and RSE respectively). ASE refers to a well-known named entity (e.g. “Paris”) while RSE refers to a complex spatial entity (e.g. “à côté de Paris” (In English: near Paris)). The taxonomy is constructed by first searching using RSE query where its ASE is a city in the country of the taxonomy. Then, a term-document matrix is constructed using the retrieved documents. After that, LSI is applied to generate the new approximated matrix which is restricted to the terms that represent ASE. Finally, the similarity is computed between the resulted ASE’s and the original, which results in a one-level taxonomy. The procedure is then repeated to build further levels. Based on this taxonomy a user’s query with geographical context can be reformulated. The results showed that the proposed method significantly improved the search’s precision.

3.2.4. Software Analysis (Code Analysis)

In [35], LSI was leveraged to detect anomalous android applications. The prevalent usage of android applications introduced many useful activities, however many anomalous applications were reported. Solutions discussed by [35] in its literature use techniques like machine learning and sand boxing but they suffer from accuracy issues (e.g. false positives). The solution proposed aims to understand the context of the permission list. To do so, a term-document is first generated. Documents represent a set of known good applications and terms are permission keywords. Then, when SVD is applied and the matrix is reduced, a common set of permissions is collected. After that, the permissions set is used to test against set of permissions for the queried application. If it matches, then it is labeled normal, otherwise it is anomalous. The paper, however, did not conduct an experiment to verify their proposed solution.

LSI was also used to detect code re-implementation. Code re-implementation is completely different from plagiarism; it is caused by the lack of awareness of existing libraries, which leads to re-implementation instead of reusing existing libraries. The resulted redundancy can range from exact copies (Type-1 clones) to replications of functionality with no code replication (Type-4 clones also known as Simions). Redundancies have a negative impact on the required effort for development and software quality.

The dataset in which LSI was applied on, according to [36], was extracted from Java code files for a specific system. The extraction included only identifiers in declaration of methods, classes and contained parameters. It captures the concepts intended by the programmers while mitigating the risk of false positives. Even though the use of LSI solely was outperformed by Aggregated Clone Detection (ACD), which achieved a precision of 52%. Combining both approaches (ACD & LSI) resulted in a precision spark that reached 76% and even 83% when both approaches results were intersected. Validating the results by the participants collected by the authors of [36] (including people working on the original system) found that results are “actionable”.

3.2.5. Document Analysis

LSI was proven effective at detecting plagiarism. Plagiarism is a spreading practice with the rapid growth of information on the web. It is described as the representations of other’s work without proper referencing. Developing plagiarism-detecting techniques requires the understanding of stylometry, which is the study of variation in literary style. Stylometry’s approach falls into two main categories, Extrinsic and Intrinsic. Extrinsic techniques perform well; however they are highly dependent on outside reference text collection (e.g. Turnitin) for detecting student essays. Intrinsic approaches on the other hand detect literary style difference based on n-grams. In [37], LSI accompanied by Stylometry are used to detect Intrinsic plagiarism over user submitted text. It is shown in the results that LSI enhanced the consistency and eliminated the noise of the text.

In [38] LSI was used to improve automatic scoring of Chinese essays. Adapting automatic scoring has been more prevalent for three factors, less human labor, less subjective factors and higher agreement rates. Nevertheless, automatic soring suffered from performance issues of Chinese text compared to English. A novel technique was proposed in [38] to overcome these issues under two main assumptions; first is that topics hidden to an essay contribute to the quality and second, an essay may include multiple topics. Therefore, topic-modeling techniques resembles a plausible remedy for the Chinese essays dilemma. A modified version of LSI was adapted to overcome its scalability issues (see section 3.3). Applying Regularized Latent Semantic Indexing (RLSI) has helped to achieve a rating agreement of 89%, demonstrating an effective solution.

3.2.6. Crime Analysis

Various attempts have been made to apply LSI in security applications as well. Fighting terror attacks is an example as proposed by [39]. At first there does not seem to be a pattern among these attacks as they vary in location and attackers. Analyzing those attacks appears to be challenging as terrorist groups appear to be dynamic. They constantly change their locations, weapons, targets and names. LSI was an advantage to provide a more profound understanding of terrorist attacks using data collected by START (Study of

Terrorism and Responses to Terrorism) from 2000-2011. The data was divided into three main chronological segments. After removing stop words, a separate term-document matrix was generated for each segment. SVD was then applied to reduce dimensions where the two first two vectors with the highest energy were clustered using K-means. Therefore, the data was clustered based on different unifying characteristics including, attack types and weapons used (i.e. attacks involving bombs and attacks involving arson). The LSI-proposed method can help achieve higher efficiency of counter-terrorism and lower response time to label terrorist groups.

LSI was used as well to assist in the problem of cyberbullying detection. It is a problem that became more prevalent and significant due to more youth having unsupervised access to the internet. It has various forms including flaming, trolling, cyberstalking, harassment, etc. according to the authors of [40]. In their paper, they used data from Formspring.com as source for rich cyberbullying data according to a previous work of theirs. In the first half of the paper, a bag-of-words language model was introduced to detect multiple types of cyberbullying. Then LSI was used to help identify additional terms that describe cyberbullying. The proposed system was described to have a high impact on giving higher scores to cyberbullying content.

Part of the team later introduced yet another system to help detecting cyberbullying [41]. However, instead of trying to detect term co-occurring with cyberbullying, a search engine is proposed. The search engine uses LSI and it does not depend on a bullying terms' dictionary. LSI has been conventionally known to be working on long well-formed text while the dataset used (retrieved from Formspring.com) is quite the contrary. As a social site, its data is usually in the form of short posts filled with misspellings, abbreviations, and unusual punctuation. Nevertheless, applying LSI managed to yield satisfying results with a precision of 55%.

Another problem LSI contributes to solve is the detection of named entities. Named entities recognition (especially names of persons) plays an important role in many intelligence and security informatics according to [42]. Most of the traditional techniques are based on techniques like phonetic similarities and edit distances. However, they usually encounter some difficulties in obtaining high precision in large datasets. The reason is that with large datasets the amount of name variants increases (e.g. Zawahiri, al Zawahiri, etc.) and hence it could include many of other individuals.

As proposed in [42], first a traditional technique with a wide acceptance threshold is used to provide a list of name variants. LSI is then used to find similar terms noting that terms are represented as vectors in LSI space. The list of name variants can be re-ranked based on LSI vector similarity. The aforementioned approach encounters some difficulties when there are multiple people with similar names doing similar activities. This occurs when the list of names is limited as in Korea and China. Hence, a refined approach was proposed to add an extra step to check for associated terms. This approach is derived from the assumption that if Name A and Name B refer to the same person then the set of terms associated with them should be similar. The refined method has proved to improve precision significantly without affecting recall.

3.2.7. Medical

LSI was correspondingly leveraged in medical applications such as liver segmentation. Computer-aided diagnosis (CAD) provide many applications for surgery planning and liver volume measurement, etc. Liver segmentation is an essential step in many CAD applications that is often done using computed tomography (CT). According to [43], automating the process of liver segmentation is important to avoid the time-consumption of the user's feedback loop required by the non-automated processes and the manual annotations of CT volumes. The liver segmentation process involves partitioning CT volume to non-overlapped sub-volumes. Given a sub-volume, it is required to apply 3D liver localization. To detect arbitrary shapes as livers, Hough transform is an effective and robust method. However, the large size of data entries resembles a great hinder for the performance of Hough transform due to its time complexity. Each CT sub-volume is represented as a set of cube features. Those feature vectors can be factorized using SVD component of the LSI, where Hough transform is then applied. The new proposed method, which uses LSI, was found to improve liver segmentation quality and helps to develop high-performance CAD applications.

Another use of LSI was in the context of annotation DBs for Gene Ontology (GO). These DBs are the repositories of human's kind biological knowledge, which keeps advancing with time. It carries information about genes and annotate each gene for a particular molecular function. However not all those annotated genes are for the corresponding biological process. This might not cause an issue if data is to be manually read by a human (i.e. DB curator or life scientist) as they are able to extrapolate the required information. However, it is more likely to conduct a simulation using a software analyzing GO graph that is constructed using annotations retrieved from the DB. In [44], it was suggested that expressing the DB as a typical term-document matrix while using LSI would solve the aforementioned issue. The problem was modeled so that genes represent terms and molecular functions represent documents. It is worth mentioning that different weighting schemes were used, other than the typical $tf - idf$ weighting scheme. Applying the dimensionality reduction was described to "capture the latent semantics and filter out the noise" [44].

3.2.8. Media

LSI has been leveraged as well in media retrieval researches. Video retrieval is one example of the potential application of LSI. The authors of [45] show how LSI can enhance video retrieval performance compared to the traditional matching algorithms used in that field. The proposed model for video retrieval contains three steps, Video segmentation, Feature extraction, and Video retrieval. First, the video is separated into multiple video shots. A shot contains multiple frames, which are assigned to it using biorthogonal wavelet transformation and L2-norm distance between every frame. Second three sets of features are extracted, including motion, color and edge density. This results in three feature vectors, which are then used to construct a feature matrix. Then LSI is applied on the feature matrix, where the result is then used to search against. Third, a query video would be processed following steps one and two where cosine similarity is used to retrieve similar videos.

The experiment conducted in [45] used a database where the similar videos for each query video was known in advance. The results reported do not show the precision of the method, however, it shows that LSI always managed to retrieve most of the similar videos.

LSI was used as well to re-rank images' tags. The need for re-ranking arises from the fact that images are usually labeled randomly. That means tags do not accurately describe the content of the image. In [46], LSI was leveraged to explore the tags' similarity as part of the re-ranking system. First, a tag-image matrix is constructed, and then the matrix is factorized using SVD where the resultant approximated matrix is used to explore the Tag-to-Tag similarity.

3.3. LSI Challenges

There are various challenges that researchers are likely to encounter while leveraging LSI. That includes, estimating the value of k, poor performance on some datasets and Scalability.

Estimating the value of k is one of the common challenges while applying LSI. In section 1.3, it is explained that SVD is the key player in LSI; it factorizes the original matrix to three matrices then a lower rank k is selected. However, selection of k is not an easy task as mentioned in [24]. If the rank is too large, the approximated matrix could end up too similar to the original, which does not solve polysemy or synonymy. If too small, it can be too dissimilar to the original and thus retain less information.

LSI also performs poorly on some selected Datasets. In [24], a sample dataset was provided where SVD failed to recognize the related documents and thus failed to address the polysemy and synonymy issue. The author also tested the performance of SVD on three different standard datasets in LSI researches, SVD failed to provide improvements over the vector space model. Another work, [26], has encountered the same problem. It found that LSI was found to perform poorly on TREC 2, 7, 8, and 2004 collections.

Poor performance on large datasets is also one of the main challenges for LSI, which is caused by the computation-intensive nature of SVD as reported by [2] and [24]. Both [2] and [24] are 25 years apart, which shows how no improvements were introduced to enhance SVD's performance. This is largely due to the orthonormal constraint mentioned in [19] and [18].

Chapter 4: Research Methodology

This chapter reflects on the problems discussed in Chapter 2. It introduces the experiments designed to tackle those issues. Discussion in this chapter can be divided into two parts. Part 1 reports on the details of experiments designed to test LSI scalability while part 2 focuses on experiment related to applying LSI on encrypted data.

4.1. Part 1: LSI Scalability[11]

One part of this work is to investigate if gensim's implementation of LSI overcomes the scalability issues discussed in section 3.3. To do so, a set of experiments were devised to perform analysis on its performance on an HPC environment on a large-scale data. The experiment measure gensim's LSI effectiveness and performance according the measurement metrics mentioned in section 4.1.4.

4.1.1. Environment

In this section, a discussion of the development environment is provided. Libraries and tools used in this part are detailed discussing their significance to this work.

4.1.1.1. Anaconda [47]

Anaconda is an open source distribution of python and R programming languages. It aims to simplify package management and deployment using package management system named Conda. It also makes it easier to create multiple virtual environment with different settings and versions on the same machine. In this work, an LSI python

[11] LSI performance and LSI's system's performance are used interchangeably in this section and through the whole document.

implementation is used (check section 4.1.1.6 for more details), thus making anaconda useful.

4.1.1.2. NumPy

NumPy is a python library introduced for scientific computing. It supports handling N-dimensional arrays and matrices along with mathematical functions to operate on them.

In this work, NumPy is utilized to apply efficient mathematical operations on array-like objects; e.g. sorting, summation, $sqrt$, generate random matrix (used in section 4.1.40), etc.

4.1.1.3. Pandas

Panda is a python library that provides data structure along with set of operation to manipulate them. In this work, panda is used to handle data like term-document matrix and relevance assessment tables which shows sorted similarities information (shown in section 5.2).

4.1.1.4. Matplotlib

Matplotlib is a 2-D plotting python library. It provides publication-quality figures and can be used in interactive applications like Jupyter (reported in the next section). It was used to plot recall-precision curves[12] and a representation of documents in LSI space (

Figure 1).

[12] Recall-precision curve is a measurement metric explained in section 4.1.4.2.1

4.1.1.5. Jupyter Notebook [48]

Jupyter Notebook is an open source web application that provides an interactive platform for over 40 different languages including python. It allows users to create and share documents over cloud services like Dropbox, GitHub, etc. or export documents as Html pages. A Jupyter document can contain live code, equations, visualizations and narrative text [48].

The descriptive nature of Jupyter documents helps to make the code clearer and easier to understand. Along with the ability to share them, it eases supervisors' engagement in the implementation side of the work, thus, it was leveraged in this work.

4.1.1.6. Gensim

Gensim is an open source python library, that was proposed in [16] as a framework (see section 3.1.4.2) that addresses scalability issues in some topic modeling algorithms including LSI. Gensim, a memory-independent library, was intentionally designed to work efficiently on large corpora. It is also an OS independent; it runs on Windows, Linux, macOS and OS X.

Gensim provides an LSI implementation that can work in both serial and distributed mode. It can handle corpora that is much larger than RAM and can be applied on a cluster of nodes for distributed mode [49]. To elaborate on the discussion presented in section 3.1.4.2, the following concepts of Gensim are presented (more details can be found in [50]).

First concept is Corpus (plural corpora), a collection of digital documents. The collection, which is also referred to as a training corpus, includes information about documents and their topic and it can be used to assign topics to new documents with no human intervention. Second concept is Vector, a feature representation of a document. Each document consists of an array of features; in the term-document matrix, for example, a feature is the number of terms that appear in a document. Third concept is Model, an abstract word that describes a transformation from one representation of a document to another. For example, the term-document matrix describes the occurrence of each term in a certain document. Applying SVD or LSI model (as described in section 1.3) would transform it to a co-occurrence matrix where documents with co-occurring term are represented closer in the vector space.

4.1.1.7. Pyro

Pyro is a python library that enables users to build applications where objects communicate over the network with minimal programming. It is used as part of gensim' implementation for DLSI. Its nameserver is also utilized in this work as described in section 4.1.3.1.

4.1.1.8. RAAD [51]

RAAD is a High-Performance Computing (HPC) cluster that is hosted by Texas A&M University at Qatar (TAMUQ). HPC refers to a large-scale aggregation of computer hardware to solve difficult problems at various fields [51]. TAMUQ has been an HPC leader in Qatar for over a decade; their goal is to help and facilitate the work of those in the Qatari research community. This decade of experiences includes management, support and maintenance of four different systems provided by the Research Computing (RC) group [51]. Those four systems are SAQR (2005 – 2008), Suqoor (2008 – 2011), RAAD

(2011 – 2016) and RAAD II (2017 – Present) [51]. In this work, only RAAD II[13] is leveraged.

RAAD II, which translates to ‘thunder’ in Arabic, is a supercomputer that was made by Cray, an American supercomputer manufacturer founded in 1972. It was co-funded by both TAMUQ and Qatar Computing Research Institute (QCRI). The RAAD II composes of 172 nodes, a total of 4,128 CPU cores, an aggregate of 22,016 GB in memory and a shared disk space of 800 TB. The full technical specifications of RAADII are reported in [52]. For scheduling and resource management, RAAD II relies on SLURM; an open-source job-scheduler for Linux and Unix-like kernels.

RAAD II would provide a perfect fit for running LSI in a distributed system. The support provided from TAMUQ to the Qatari research community will help granting access for a powerful computing hardware along with extensive support. For this reason, RAAD II was selected for this work.

4.1.2. Datasets

In our experiments, two different datasets will be leveraged; Wikipedia and Medline dataset [53]. This section provides a brief description of each dataset and its role in this work.

4.1.2.1. Medline

Medline is a dataset of medical articles. The School of Computing Science in the University of Glasgow provides the version used in this work. It was specifically designed to for IR-related tasks, which implies that an evaluation analysis can be carried on (see

[13] For simplicity RAAD will be used to refer for RAAD II in the upcoming sections.

more in sections 4.1.4.2 and 4.1.3.2.2). The dataset contains three files; *MED.ALL* which contains a list of 1033 articles, *MED.QRY* that contains a list of 30 possible queries and *MED.REL* containing a list of all relevant documents for each query.

4.1.2.2. Wikipedia

Wikimedia frequently provides a dump of Wikipedia articles, which were used by the genism team. The gensim team provided a getting-started guideline to apply LSI on local machine using the Wikipedia dataset as an example [54] and they also leveraged it in their paper [30]. In this work, a dump released on 13 April 2017 is being used; however, the experiment can be conducted on any release. The size of the dump is 12.8 GB in size and all the articles (4,227,933 articles) are available in one xml file.

While Wikipedia dataset is not a comparable scale to data in the context of web search (4 million dataset), we consider a scale of data that is owned by a single entity and stored on an HPC platform provided by a cloud computing service provider.

4.1.3. Experiments

4.1.3.1. LSI Performance

4.1.3.1.1. Introduction

The overall goal of the experiments designed in this section is to test if LSI can scale over large datasets. The LSI implementation used in this set of experiments is the gensim's LSI discussed in Gensim in sections 3.1.4 and 4.1.1.6. These experiments are conducted on RAAD using both modes of gensim's LSI, serial and distributed.

The serial mode requires only one node and it runs in sequential steps. Dataset reading is the first step; preprocessing (if any) usually follows the reading step or is integrated with it (further details are provided in the Measurement Metrics section). Secondly the corpus is generated, which is a term to describe a genism collection [50]. Once a corpus is created from an existing dataset, it is ready for further processing using genism. It records information of dataset document and features (distinct words after preprocessing), number of non-zero entries in the term-document matrix, and a map between all words and their integer ids (i.e. dictionary) [55]. In practice, the dictionary is stored separately even though it is part of the corpus. Both corpus and dictionary are then used to generate the LSI model.

The distributed mode follows the same steps; however, it requires four different objects running on different machines representing the cluster. First is the pyro nameserver, a tool to help keep track of objects running on the network and it also assigns logical names to those objects instead of exact IPs for convenience. Second is the worker, a class implemented by gensim that handles the actual computation in distributed LSI (DLSI). Third is the dispatcher, a "job scheduler in charge of worker synchronization" [49]. It prepares documents' chunks and assigns them to workers. The aforementioned objects are the essential entities of the cluster. Fourth and last object is a mere python (and gensim) implementation of DLSI.

4.1.3.1.2. LSI on RAAD

RAAD uses slurm as its workload manager as aforementioned. When a job is submitted, it allocates the job to the next available node. A job in this context refers to the code of a single object (e.g. worker, dispatcher, etc.) that is submitted to RAAD. To run DLSI, the nameserver is first submitted to run on one node which is randomly selected by slurm. Secondly, logical worker(s) are submitted where the physical node can be either manually selected or left to the queue to decide. The dispatcher is then submitted followed that the DLSI code.

In this work, the purpose is to check if LSI can be scaled up, thus, various setups were devised to test that. The Wikipedia dump is large, it takes a relatively long time to finish and that makes it difficult to debug any possible issues. Therefore, a decision was made to start from a small sample and build up for the original size (4 million document).

Wikipedia divides its full dump to smaller files of arbitrary sizes as can be found in [56]. In this work, a file of size 300 MB (113,550 articles) was selected. Then the gensim's function RepeatCorpus was applied on that dataset to generate datasets of the following sizes, 500k, 1m, 1.5m, 2m, and 4,227,933 articles[14].

For each dataset size, LSI is run in both modes, serial and distributed mode. The distributed mode is run using on 2, 4, 6 and 8 workers. Each two workers run on the same physical machine to follow the same setup of [49].

It is important to note that while using RepeatCorpus, the dictionary's (word-id mapping) size does not change as the list of work is the same. Thus, LSI is applied on both the original 4m dataset and on the repeated one to check the dictionary's size effect. In addition, the chunk size was reported to affect LSI's performance [30], thus different chunk sizes were used to test its effect.

In addition to testing the effect of both the dictionary and chunk size on LSI performance, the setup discussed earlier helps to study further points. It helps to study how

[14] Same size the Wikipedia dataset (section 4.1.2.2)

gensim’s LSI scale up across different sample sizes and how it scales up across different number of worker (and in turn, different number of machines). This help to determine the efficiency of the genism library. The library’s performance is also compared to the baseline discussed in Baseline 1: Distributed LSI on Wikipedia.

4.1.3.1.3. Baseline 1: Distributed LSI on Wikipedia

In [30], a distributed solution for LSI was proposed to handle large datasets as formally. The dataset utilized in the experiment was the Wikipedia dataset mentioned in Wikipedia.

The dataset was represented in a 100,000 x 3,199,665 sparse matrix and it had 0.5 billion non-zero entries (0.15% density). The dataset was clipped to 100k features (most frequent words) and the rank k was set to 400.

The various experiments reported in [30] were designed to evaluate various variations of DLSI before choosing the default variation implemented in gensim. In this work we consider the default variation to be the baseline. The experiment was performed on 2.0 GHz Intel Xeon workstation with 4GB of RAM and 4 nodes (8 worker) with CPU time is 1h 41m. It was reported on gensim’s official site as well [49].

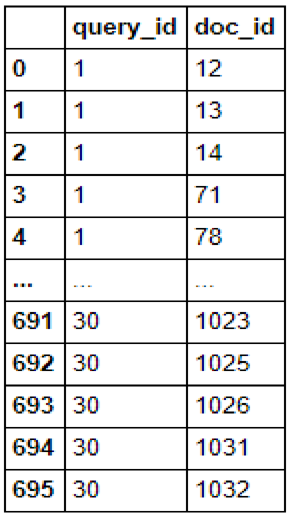
4.1.3.2. LSI Effectiveness

The overall goal of the experiments designed in this section is to test the impact of distributed on gensim’s LSI effectiveness. In this case effectiveness is measured by the average precision as discussed in section 4.1.4.2. The dataset utilized in this section is the Medline dataset, which helps to calculate the average precision as formerly mentioned.

4.1.3.2.1. Experiment

The experiments in this section follows the same steps reported in the LSI Performance adding an extra step of relative assessment to measure the effectiveness.

To do so, first, a new index reference is prepared through applying the Similarity class [57]. The Similarity class job is to divide an index into sub-indexes (shards), this is mostly helpful in case the entire index cannot be fit in memory. The output of the similarity class (i.e. new index reference) can be used to calculate the similarity between a query and a document in the index. Second, a relevance table is constructed using MED.REL. The relevance table contains a list of all query ids in MED.QRY a long with their relevant document ids as shown in the figure below.



	query_id	doc_id
0	1	12
1	1	13
2	1	14
3	1	71
4	1	78
...	...	...
691	30	1023
692	30	1025
693	30	1026
694	30	1031
695	30	1032

Figure 5: A snapshot from relevancy table generate from Medline dataset

Third, a list of Medline queries is read from MED.QRY file, while the similarity for each of them against all Medline documents is calculated using the index. The list of similarities is then sorted in a descending order from the most relevant document to the least relevant.

Both the table of relevance and list of similarities can be used to calculate the precision and recall for each query.

4.1.3.2.2. Baseline 2: Medline

The experiment detailed above is applied in both serial and distributed mode. The average precision is calculated in both cases to measure the effectiveness. The comparison between both modes helps determine the impact distributed mode has on LSI and whether it maintains the same level of effectiveness.

4.1.4. Measurement Metrics

Throughout the three scenarios, two main metrics will be analyzed, performance and evaluation. Both metrics will be later utilized to compare an experiment's outcome against one of benchmark solutions discussed in 4.1.3.

4.1.4.1. Performance

The performance metric is a measurement of CPU time taken by a machine to build an LSI model. Further details on how the CPU time was consumed shall be provided along each experiment's results' analysis.

4.1.4.2. Effectiveness/Accuracy

In IR, evaluation metrics plays a key role to determine the effectiveness of an IR system. Those metrics usually "revolve around the notion of relevant and nonrelevant documents" [1] where the relevancy is a measurement of the satisfaction of user needs which are expressed in their query.

Measuring the effectiveness of a system can then help to decide on the most efficient pre-processing steps. This includes removing stop-words, stemming, etc. It also helps to determine whether any of the proposed experiments are valid or not.

4.1.4.2.1. Recall-Precision curve

Traditional effectiveness measures like precision and recall are more suitable for set-based measures [1]. In this case, these measures are applied on a set of un-ordered document. However, in this work, the retrieved set of documents are in order, where the top-k documents represent the most-k relevant documents, similar to a search-engine retrieval list.

For an ordered-set, a precision-recall curve can be plotted using the calculated data for precision and recall. The curve shows both precision and recall values calculated in each document.

4.1.4.2.2. Interpolated Average Precision

The recall-precision curve provides an easy-to-interpret method to judge how the effectiveness of the system varies while moving further in the retrieved list. However, it cannot be used to compare performance across different queries. For that purpose, interpolated average precision is used. The interpolation values are calculated at 11 standard recall levels (100%, 90%, 80%, ..., 10%, 0%). It can be plotted in an easy-to-interpret graph and also it can be used to calculate a single value measure; 11- point Interpolated Average Precision [58].

4.1.4.2.3. Average Precision (AP)

MAP works similarly to the Interpolated Average Precision; it helps to compare effectiveness across different queries and generate a single-value measure, Mean Average Precision (MAP).

4.1.4.2.4. Other Measurements

Other measurements include Discounted Cumulative Gain (DCG) and F1-measure. DCG is more suitable in the case of a graded relevancy measurement (e.g. 5=very relevant, 4=relevant, etc.) which is out of the scope of this work. Besides, F1-measure is more suitable for professions that highly value recall figures like paralegals and intelligence analysts [1].

For this work both the Interpolated Average precision and MAP can be leveraged, however, MAP is the "most standard among the TREC community[15]" according to [1].

4.1.5. Datasets Pre-processing

The preprocessing stage is vital to validate the articles in the dataset and prepare it for processing. It can include a step to prepare the dataset to be readable by the gensim library and it can include traditional IR preprocessing steps like tokenization and stop-

[15] Text REtrieval Conference (TREC) is a conference that aims to support research within the information retrieval community (see here: http://trec.nist.gov/overview.html)

words removal. The aim of traditional pre-processing steps is to optimize the effectiveness of the system. However, this work is concerned about the relative performance across the implemented experiments, thus, those pre-processing steps were neglected. This section describes the details of the applied pre-processing steps on each dataset.

4.1.5.1. Medline

Gensim provides various classes to read a collection and build a corpus, the one used in this work is *text_corpus*. The *text_corpus* class uses the method *get_texts* to read a given dataset. In this work, the *get_texts* method was updated to fit the Medline articles' format. Defaults pre-processing settings of *text_corpus* is used.

4.1.5.2. Wikipedia

Gensim provides a *wiki_corpus* class that provides the same functionality of *text_corpus* for the Wikipedia collection. In this work, *wiki_corpus* was utilized along its default pre-processing settings

The *wiki_corpus* class applies a set of different methods that pre-process the collection and remove non-necessary parts. Following is a list of functionalities applied to clean the Wikipedia collection.

4.1.5.2.1. Page Extraction

One of the very first methods applied by the *wiki_corpus* is *extract_pages*. The method takes the Wikipedia collection as an input and returns a triple of *title*, *content* and *pageid*.

4.1.5.2.2. Filtering

The content of each page is then passed to the *filter_wiki* method. It generates a utf-8 encoded string that is then passed to the *remove_markup* method. Wikipedia-related markups are then removed or processed, this includes removing comments, removing footnotes, re-structuring tables and simplifying links.

4.1.5.2.3. Tokenization

The previous steps' output a stream of filtered pages (i.e articles). Each page is then passed to *tokenize* method. It takes the filtered pages as an input and outputs index-ready tokens, ignoring words shorter than 2 or longer than 15 characters.

After that, a Wikipedia corpus is ready to be used to generate the LSI model.

4.2. Part 2: LSI on Encrypted Data

The second part of this work focuses on proposing a framework/protocol that allows to perform search functionality using LSI on a cloud server while obscuring the data from it. To do so, multiple experiments with various settings were devised and tested. In this section, we report on the adapted environment and tools. Then we list the leveraged datasets followed by the details of the carried-on experiments. Details on measurement metrics and dataset preprocessing are also provided.

4.2.1. Environment

The development environment in this part is largely similar to the environment discussed in the former part. However, LSI was implemented from scratch here (see section 4.2.3) and run on a local machine, thus both gensim and RAAD were not used.

A new tool that was used in the part is *Sklearn*. It is a machine learning library for Python. It provides multiple algorithm for classification, regression, clustering and others. However, for this work, *Sklearn* was utilized to generate the term document matrix and apply SVD on it. For this purpose, both classes *CountVectroizer* and *randomized_svd* were used. More details can be found in the experiments section.

4.2.2. Datasets

4.2.2.1. Proof of Concept

In this section a simple dataset constructed by [4] and shown in Table 1 is used. The dataset was initially introduced to show how LSI works and thus it aligns with the aim of this part of the work; showing how LSI works on encrypted data. It mainly used as a proof of concept for our proposed protocol. This dataset is used as a proof of concept to illustrate the effectiveness of our methods.

4.2.2.2. Medline

As a test collection, Medline can help evaluating our methodology on a real test collection that is used in research area. While the following experiments represent a development process to design a practical protocol, Medline is only used in the last one (i.e. finalized version of the protocol). Using Medline, we compare both the performance and effectiveness of our proposed protocol.

4.2.3. Experiments

4.2.3.1. Overview

In this set of experiments, the cloud service provider is assumed to be the attacker, thus the aim is to hide the data from it. To achieve that, we try to build a secure system using security via randomized splitting, a technique that secures data by splitting it into smaller units with the help of a random variable and distributes them to separate locations. Therefore, the system uses multiple cloud servers assuming they do not collude or have no method of communication (e.g. different cloud service providers). For example, if we have the following two variables

$$A = 7$$

$$R = 3$$

Assuming A is the variable, we want to hide and R[16] is the random variable, randomized splitting will generate two versions of A

$$A_1 = A + R = 7 + 3 = 10$$

$$A_2 = -R = -3$$

Each version of A is then stored at a different cloud server, and if both adhere to the protocol while having no method of communication, they cannot interfere with the original value of A. Since computers, as deterministic systems, cannot really generate a truly random number [59], therefore, security of the system is conditional based on the random function. Those pseudo-random numbers are sufficient for most applications and that is why a system that is secured by random splitting can be – to a high extend- assumed to achieve unconditional security.

The goal in these experiments is to maintain the ability to search while using random splitting. The baseline experiment described in section 4.2.3.2 retrieved the human-computer interaction documents at the top, thus if that is achieved, the outcome is

[16] In some cases, two different random matrices are used in the same experiment. In this case a different symbol might be used like r

satisfactory.

No actual cloud servers were used in these experiments; however, they are simulated in the variable names. So, for two variables A_1 and A_2, they are hosted by $S1$ and $S2$ respectively. For each experiment, similarity is calculated twice, both before and after vector's length normalization. Results are then discussed and compared in case any difference in the results occurs.

The upcoming sections discuss the setup details and objective for each conducted experiment along with an overview graph that depicts the flow of the experiment.

4.2.3.2. Baseline

The experiment designed in [4] and discussed in section 1.4 is selected as the baseline solution for this part. It is considered as a standard dataset to describe LSI's functionality which aligns with the goal of this part. Leveraging this experiment also makes it easy to compare results.

For this part genism's LSI was not utilized; instead it was manually written from scratch. While genism implantation is more efficient, memory friendly and can be scalable as discussed in section 4.1.3.14.1.1.6; we require high control on the implementation for later modifications. Gensim, as an open source project, may provide the possibility for alternation, yet it is a huge project and modifications on it could require more effort than what the scope of this thesis allows.

First step in this implementation is to generate the term-document matrix A, that is achieved using class CountVectorizer of sklearn library. Sklearn also provides the functionality to factorize A to three matrices as explained in 1.3 using the class randomized_svd. The output three matrices are already in a lower rank, so their multiplication is A_k.

$$U, Sigma, V^T = randomized\_svd(A, n\_components = 2)^{17}$$

Equation 5: Factorize the matrix A using sklearn's *randomized_svd*

$$A_k = U.dot(Sigma).dot(V^T)$$

Equation 6: Calculation of A_k using the output of *randomized_svd*

After that, a query vector q is generated and then transformed to the same space as A_k. It is important to note that the transformation uses two of the three factor matrices composing A_k; U and $Sigma$.

$$q_k = q.transpose().dot(U).dot(Sigma.transpose())$$

Equation 7: Calculation of q_k using the output of *randomized_svd*

qk is then used to calculate $sim(q_k, A_k)$; the cosine similarity between the query and the matrix A_k as discussed in 1.3. This similarity is the baseline that the following

[17] *n_components: Number of singular values and vectors to extract. In this case number of component is two.*

experiments compare against.

Since we are using two datasets, the baseline is applied in both.

4.2.3.3. Experiment 1

Our first goal was to test the ability of hiding the matrix A through random splitting into two servers $S1$ and $S2$ without affecting LSI effectiveness. That means the experiment will retrieve the c-coded documents at the top. If successful, it can be interpreted that only generating the term-document matrix (occurrence matrix) has to be performed at the client side. Thus, generating the LSI model (including the computationally intensive SVD) will be performed at the cloud. The following two sections discuss the details of the experiment while section 0 depicts a graph overview of the experiment.

4.2.3.3.1. Prepare Matrix A

First step in the experiment is to generate the term-document matrix A, then SVD is applied on A which yields three different matrices as explained in 1.3 and shown in Equation 5. Those three matrices are then used to calculate A_k as shown in Equation 6. After that, R is used to split A into two values assumed to be stored on two different servers; $S1$ and $S2$.

$$A_1 = A + R$$

$$A_2 = -R$$

Equation 8: Splitted A

After that, each server applies SVD on its version of A to calculate A_k as follows.

$$U1, Sigma1, V^T{}_1 = randomized\_svd(A1, n\_components = 2)$$
$$U2, Sigma2, V^T{}_2 = randomized\_svd(A2, n\_components = 2)$$

Equation 9: Applying SVD on both servers

Then A_k for each server is calculated.

$$A_{1k} = U1.dot(Sigma1).dot(V^T{}_1)$$
$$A_{2k} = U2.dot(Sigma2).dot(V^T{}_2)$$

Equation 10: Calculating A_k on each server

4.2.3.3.2. Preparing Query q_k

As previously mentioned in section 4.2.3.2, the query q is transformed into the same space as A_k using U and $Sigma$ shown in Equation 7. In this experiment q_k is calculated for each value of A_k as each is hosted in a different cloud server. The query is first sent to the cloud servers and then calculated as following.

$$q_{1k} = q.transpose().dot(U1).dot(Sigma1.transpose())$$
$$q_{2k} = q.transpose().dot(U2).dot(Sigma2.transpose())$$

Equation 11: Calculating q_k on each server

4.2.3.3.3. Similarity Calculation

The similarity between each query and its counterpart is then calculated; $sim(q_{1k}, A_{1k})$ and $sim(q_{2k}, A_{2k})$. Both similarities are then added up and compared to $sim(q_k, A_k)$ as the baseline.

4.2.3.3.4. Experiment Overview

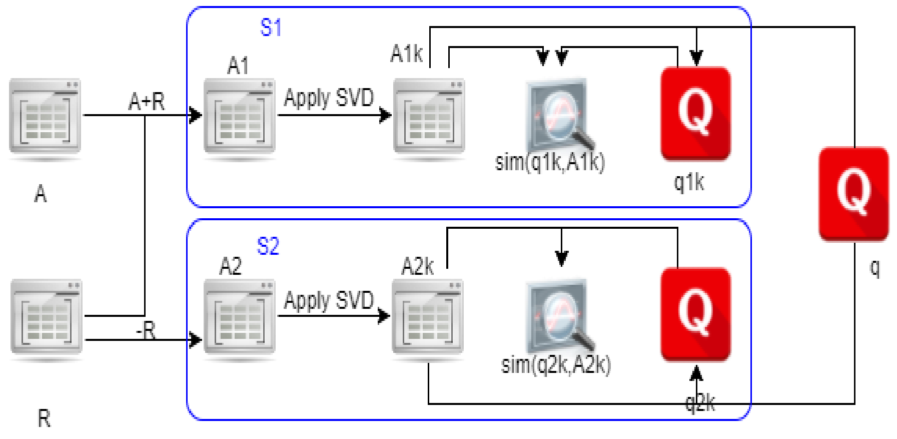


Figure 6: LSI over Encrypted data – Experiment 1, Dividing Matrix *A*

The graph above shows the flow of both *A* and *q* according to the steps described in former section.

4.2.3.4. Experiment 2

This experiment was proposed as Experiment 1’s results were not conclusive as will be shown later. The premise behind this experiment is to hide A as well, however, it is done differently as it is discussed in the following section. Similar to the previous experiment, if successful, this provides a method to hide A from the cloud server while forwarding most of the heavy computation to the remote cloud server. Both sections 4.2.3.3.2 and 4.2.3.3.3 discuss how the system can be queried and how to calculate similarity while section 4.2.3.4.4 depicts an overview graph of the experiment.

4.2.3.4.1. Preparing Matrix A

Contrary to what is done in experiment 1, A is not divided using a random matrix, however, SVD is first applied as shown in Equation 5. The random matrix r is applied to only the matrix $Sigma$ as shown below.

$$Sigma1 = Sigma + R$$

$$Sigma2 = -R$$

Equation 12: Splitting Sigma

Both values of $Sigma$ are then used to calculate two different A_k as the following equation shows (all these steps are assumed to be performed locally).

$$A_{1k} \; = \; U.dot(Sigma1).dot(V^T)$$

$$A_{2k} \; = \; U.dot(Sigma2).dot(V^T)$$

Equation 13: Calculating both versions of A_k locally

4.2.3.4.2. Preparing Query q_k

When a query q is formed and sent to a server, it is transformed based on the A_k variable it hosts. The following equation describes the transformation process.

$$q_{1k} = \; q.transpose().dot(U).dot(Sigma1.transpose())$$

$$q_{2k} = \; q.transpose().dot(U).dot(Sigma2.transpose())$$

Equation 14: Calculating q_k on each server

4.2.3.4.3. Similarity Calculation

The similarity between each query and its counterpart is then calculated; $sim(q_{1k}, A_{1k})$ and $sim(q_{2k}, A_{2k})$. Both similarities are then added up and compared to $sim(q_k, A_k)$ as the baseline.

4.2.3.4.4. Experiment Overview

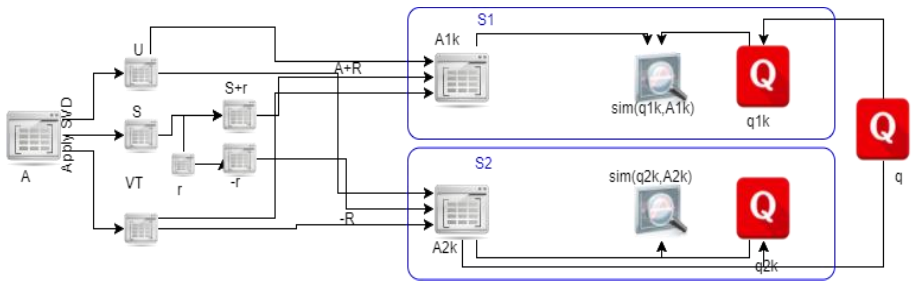


Figure 7: LSI over Encrypted data – Experiment 2, Dividing Matrix A after applying SVD

Similar to the previous figure, the figure above shows the flow of both A and q according the discussion provided in the previous sections.

4.2.3.5. Experiment 3

The previous two experiments did not yield satisfying results; hence, a new alternative is proposed. This experiment shifts its goal from hiding the term-document matrix A to hiding the approximated matrix A_k; thus, requiring that SVD is performed on the data owner's local machine.

In this the premise that the A_k can be separated into two servers; $S1$ and $S2$, and then yield the same results when queried is tested. If successful, this means that A_k can be hidden from cloud service without affecting LSI effectiveness. Following three sections discuss the details of the experiment while section 4.2.3.5.4 depicts a graph overview of the experiment.

4.2.3.5.1. Preparing Matrix A_k

First step in the experiment is to generate the term-document matrix A, then SVD is applied on A which yields three different matrices as explained in 1.3 and shown in Equation 15.

$$U, Sigma, VT \ = \ randomized\_svd(A, n\_components = 2)$$

Equation 15: Applying SVD to A with rank k=2

Those three matrices are then used to calculate A_k.

$$A_k = \ U.dot(Sigma).dot(V^T)$$

Equation 16: Calculate A_k

Each column (i.e. document) in $A_k$is then normalized where R is used to separate $A_{k(normalized)}$ into two values assumed to be stored on two different servers; $S1$ and $S2$.

$$A_{1k(norm)} = \ A_k + R$$

$$A_{2k(norm)} = \ - R$$

Equation 17: Split $A_{k(normalized)}$

4.2.3.5.2. Preparing Query q_k

As aforementioned the query vector q should be transformed to the same space as the matrix A_k. The transformation is shown in Equation 7.

4.2.3.5.3. Similarity Calculation

After that, the similarity between q_k and $A_{1k(norm)}$ and $A_{2k(norm)}$ is consequently calculated; $sim(q_k, A_{1k(norm)})$ and $sim(q_k, A_{2k(norm)})$. Both $sim(q_k, A_{1k(norm)})$and $sim(q_k, A_{2k(norm)})$ are then added up and compared against $sim(q_k, A_k)$ as the baseline. Results are discussed in the next chapter.

4.2.3.5.4. Experiment Overview

The figure below shows a flow chart of the experiment discussed above.

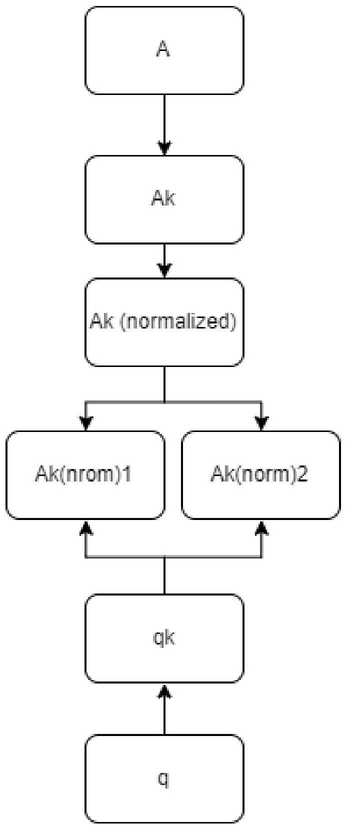


Figure 8: LSI over Encrypted data – Experiment 3, Dividing the matrix A_k

4.2.3.6. Experiment 4

The results of the previous experiment showed promising results (details of the results discussed in the next chapter) in hiding A_k; thus, this experiment follow similar steps to test if the query q_k can be split and then each split is used to query a different server. If successful, this means that q_k can be hidden from cloud service without affecting LSI effectiveness. Therefore, both experiments can be later combined to hide both A_k and q. Following three sections discuss the details of the experiment while section 4.2.3.6.4 depicts a graph overview of the experiments.

4.2.3.6.1. Prepare Matrix A_k

In this experiment A_k is neutralized as a factor so it is not divided.

4.2.3.6.2. Prepare Query q_k

The steps to calculate q_k are the same as described in section 4.2.3.5.2. However, an additional step is added to first normalize the query q_k to $q_{k(normalized)}$ and then divide it to two different variables; $q_{1k(norm)}$ and $q_{2k(norm)}$. To do so, a random matrix r of the same size as $q_{k(normalized)}$ is first generated. Then r is used to separate it as following.

$$q_{1k(norm)} = q_{k(orm)} + r$$

$$q_{2k(norm)} = -r$$

Equation 18: Split $q_{k(normalized)}$

4.2.3.6.3. Similarity Calculation

After that the similarities between each of $q_{1k(norm)}$ and $q_{2k(norm)}$ and A_k are calculated; $sim\left(q_{1k(norm)}, A_k\right)$ and $sim(q_{2k(norm)}, A_k)$. Both similarities are then added up and and compared against $sim(q_k, A_k)$ as the baseline. Results are discussed in the next chapter.

4.2.3.6.4. Experiment Overview

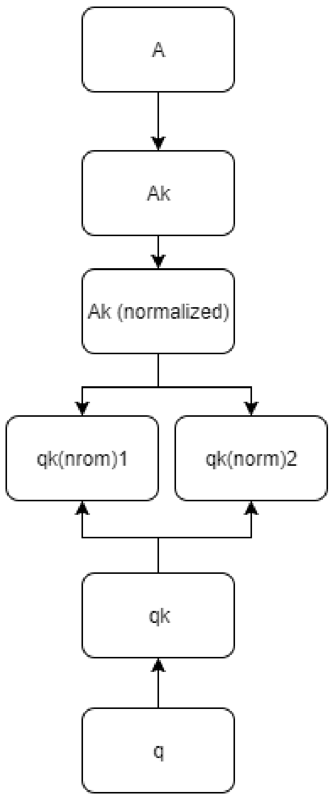


Figure 9: LSI over Encrypted data – Experiment 4, Dividing the query q_k

4.2.3.7. *Experiment 5*

The previous two experiments managed to split both A_k and q_k separately, which shows that hiding both values from the cloud is possible. This experiment aims to combine the work of both experiments, and if successful this means both A_k and q_k can be hidden at the same time without affecting LSI effectiveness. In this experiment four different cloud servers are assumed to be used, the role of those servers is discussed in the rest of the section.

4.2.3.7.1. Prepare Matrix A_k and Query q_k

The steps performed here are copied from the former two experiments. While preparing A_k is taken from section 4.2.3.5.1 that shows how A_k is hidden from the cloud, steps for preparing q_k were taken from 4.2.3.6.2 which shows how to hide q_k. At the end of these steps four values are calculated $A_{1k(norm)}$, $A_{2k(norm)}$, $q_{1k(norm)}$ and $q_{2k(norm)}$

4.2.3.7.2. Similarity Calculation

In this experiment four different similarities are calculated. This means that the similarity between each divided query and each divided matrix is calculated; this includes $sim(q_{1k(norm)}, A_{1k(norm)})$, $sim(q_{1k(norm)}, A_{2k(norm)})$, $sim(q_{2k(norm)}, A_{1k(norm)})$ and $sim(q_{2k(norm)}, A_{2k(norm)})$. After that, all four similarities are added up and then compared to the baseline.

The experiment is then repeated on the Medline dataset on a set of 30 different queries.

4.2.3.7.3. Experiment Overview

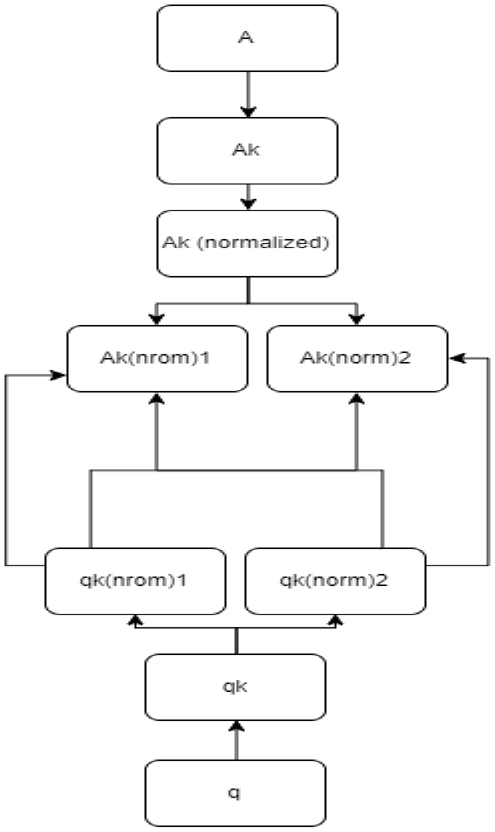


Figure 10: LSI over Encrypted data – Experiment 5, Dividing both A_k and q_k (Two Servers)

The figure above shows the flow of data in this experiment; however, it still contains a major flow. If we assume that both $A_{k1(norm)}$ and $A_{k2(norm)}$ are each stored on a different server such that one server cannot learn both values of $A_{k(norm)}$. We still face an issue that each server will learn both variations of $q_{k(norm)}$. Therefore, we decided that each variation of $A_{k(norm)}$, will be stored on two different servers as shown in the figure below.

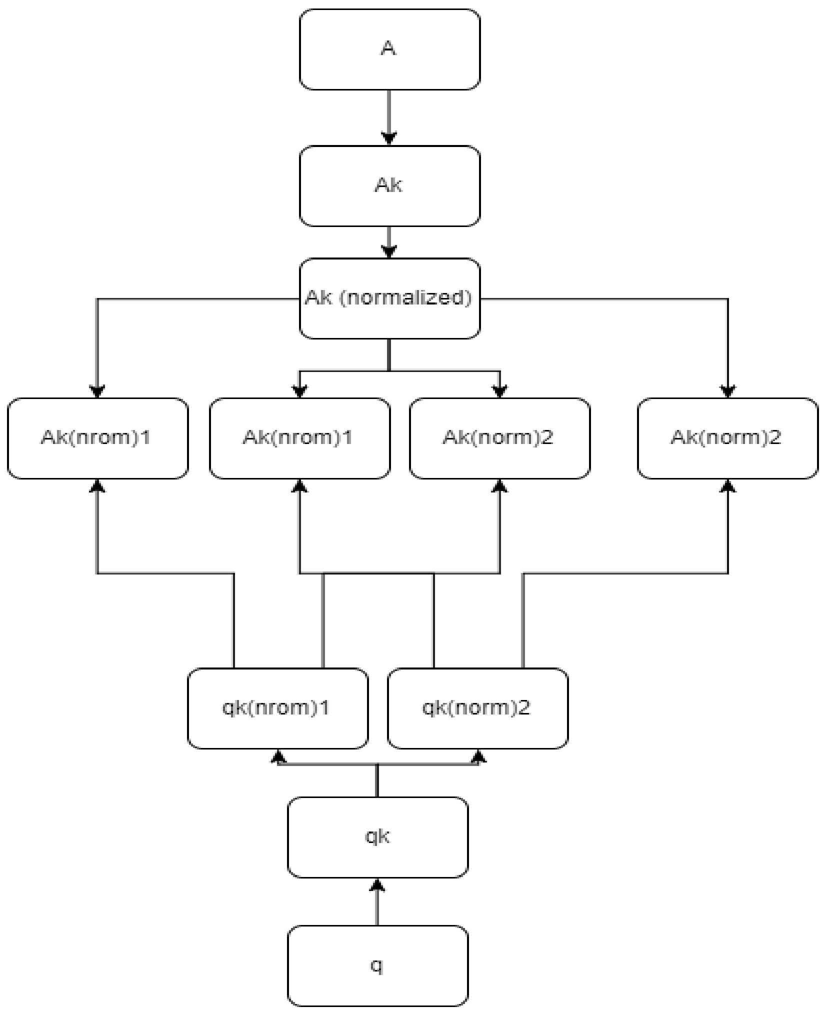


Figure 11: LSI over Encrypted data – Experiment 5, Dividing both A_k and q_k (Four Servers)

4.2.4. Measurement Metrics

4.2.4.1. Proof of Concept

To determine if a proposed protocol is valid or not, the experiment should show comparable results to the baseline. For each experiment, a ranked list of the results is shown and compared to the output of the baseline (details can be found in 4.2.3.2, 4.2.3 and 5.2).

4.2.4.2. Medline

After the simple dataset is used as a proof of concept, experiment 5 is repeated on the Medline dataset. MAP is used to evaluate the effectiveness of experiment 5 vs the baseline, and CPU time is used to evaluate the performance.

4.2.5. Datasets Preprocessing

For this challenge, no preprocessing steps were performed. The reason is that this challenge aims to compare relative effectiveness in each experiment against the baseline solution. Similar effectiveness results mean that the proposed framework is valid and can be applied in practical environment.

Chapter 5: Experimental results

5.1. Part 1: LSI Scalability

This section reports the results of experiments discussed in section 4.1.3.

5.1.1. LSI Effectiveness

In this section, we report the MAP of gensim’s LSI in both serial and distributed mode. As formerly mentioned, Medline dataset was leveraged as a test collection to allow us to measure the effectiveness.

The figure below shows the APs at different 30 Medline queries in both serial and distributed mode. Overall DLSI achieved a higher MAP (0.20) compared to (0.186) in the case of serial mode. The full list of APs can be found in APPENDIX I: LIST OF ALL RAAD EXPERIEMENTS.

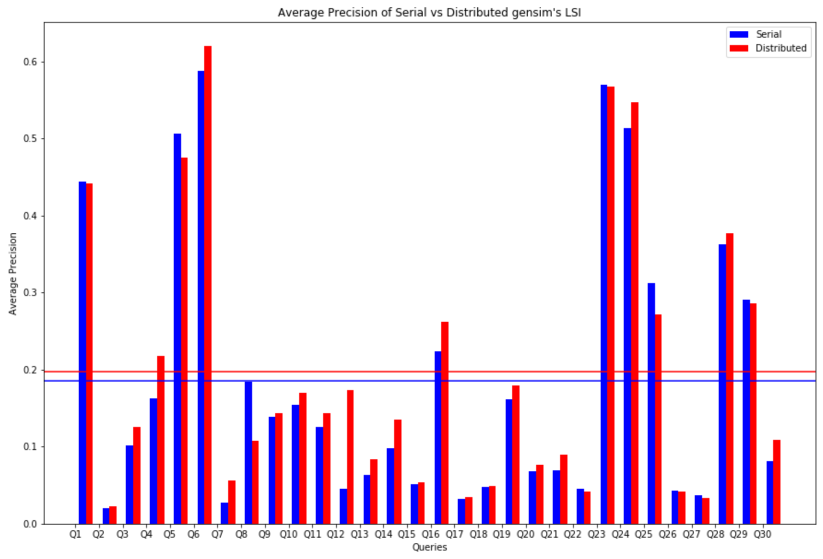


Figure 12: Average Precision of Serial vs Distributed gensim's LSI

5.1.2. LSI Performance[18]

In this section, different graphs are presented to depict LSI's performance with respect to several factors. The experiments were conducted on RAAD using the large-scale Wikipedia dataset to test gensim's LSI ability to scale. The full list of tables reporting all figures of conducted experiments can be found in APPENDIX I: LIST OF ALL RAAD E.

5.1.2.1. Serial vs. Distributed LSI

The figure below shows the difference in performance between both serial and distributed LSI across different data sizes. The distributed mode shown used 8 workers on 4 physical nodes (2 workers per node).

[18] All reported experiments use a cluster size of 20000 unless otherwise mentioned

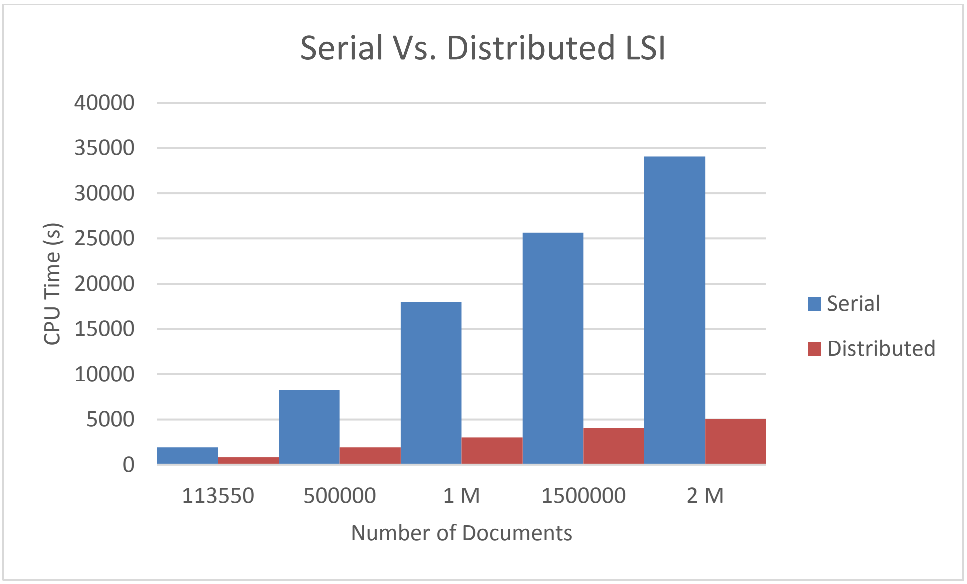


Figure 13: Difference in Performance between Serial and Distributed LSI Across Different Datasets Sizes

Conveniently, the CPU time increases as the dataset gets larger. However, it can be noted that CPU time increases much faster in the case of serial LSI. The graph also shows that distributed LSI performed better in call cases.

5.1.2.2. Number of Workers Effect on Performance

The graph below shows the difference in performance of DLSI while using different number of workers.

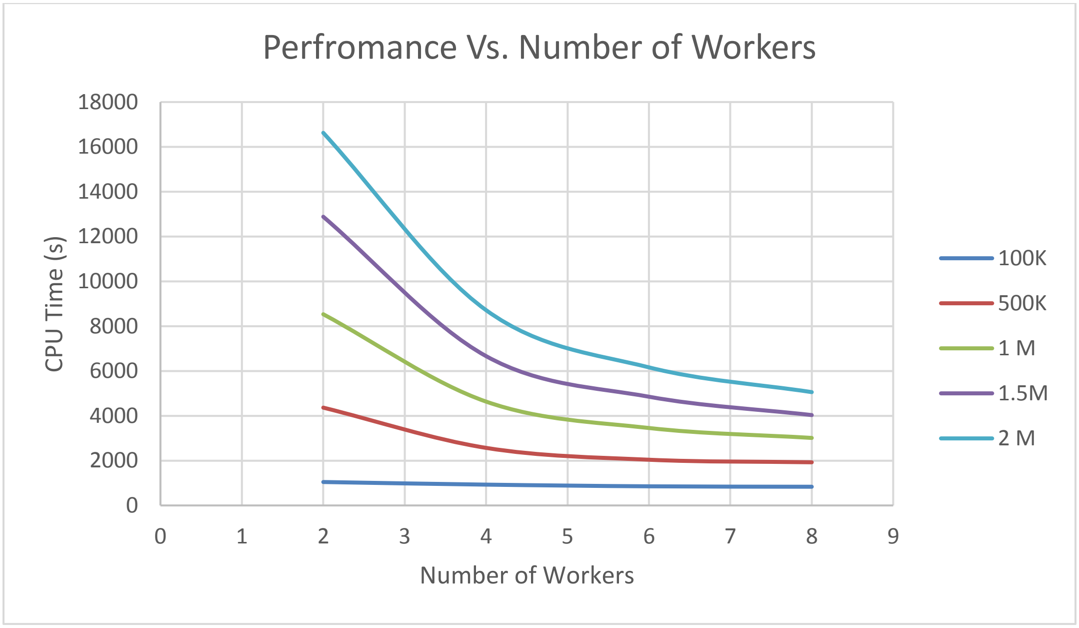


Figure 14: Number of Workers effect on CPU time

Figure 14 shows how adding more workers in distributed LSI always boost the performance. It also shows that the eventually the performance converges which implies that adding more workers is not always needed to enhance the performance.

5.1.2.3. Chunk Size Effect on Performance

The figure below shows the difference in performance of DLSI while using different chunk sizes. This is to test the claim that larger chunks lead to faster performance (see section 3.1.4.5). The experiments are run using 8 workers on 500k documents.

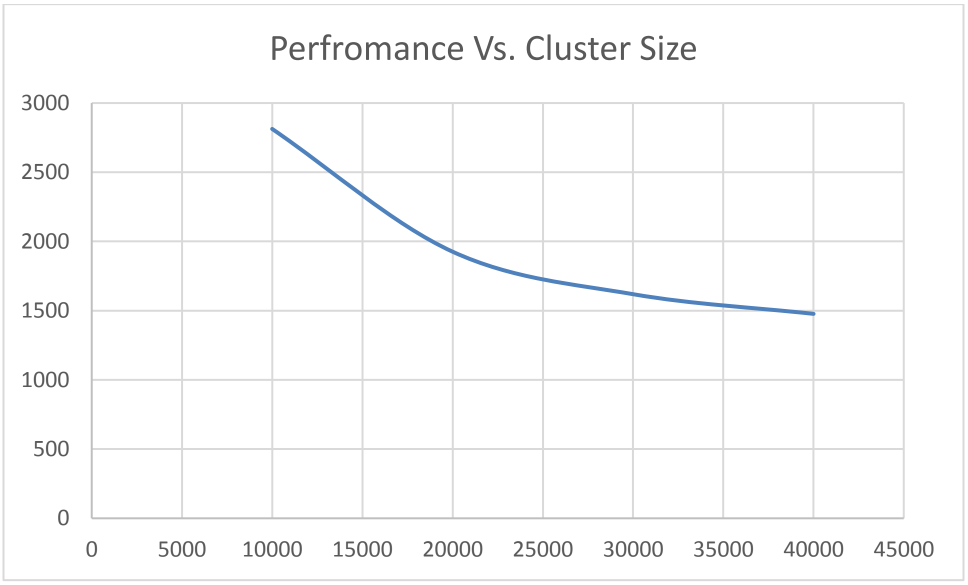


Figure 15: Chunk size effect on CPU time

Figure 15 shows how the performance is proportionally related to the chunk size, the faster LSI is.

5.1.2.4. Dictionary Size Effect on Performance

As mentioned in the previous chapter, DLSI is applied on the original 4m dataset and the repeated one to analyze the effect of the dictionary size. The figure below shows CPU time in both cases.

Table 3: Difference in Performance between Repeated and Non-Repeated Datasets in Serial Mode

SERIAL LSI	**TIME**				
	d	h	m	s	time (s)
REPEATED	0	20	8	8	72488
NON-REPEATED	2	20	55	50	248150

Table 4: Difference in Performance between Repeated and Non-Repeated Datasets in Distributed Mode

DLSI	**TIME**				
	d	h	m	s	time (s)
REPEATED	0	4	52	8	17528
NON-REPEATED	0	7	50	0	28200

Both tables show a similar trend; dictionary size affect the performance of LSI. This shows how LSI's performance could vary depending on how diverse the dataset is.

5.1.3. Discussion

Gensim's LSI's effectiveness did not see a drop when distributed mode was leveraged. On the contrary, the effectiveness was slightly higher. This shows that the algorithm lives up to its promises by providing a production-ready implementation of distributed LSI.

The difference in performance of LSI in both modes, serial and distributed, shows that gensim's DLSI is able scale over large datasets. This is also confirmed when further distributing the cluster by adding more nodes. The figure clearly shows that increase the number of nodes boost the performance at first until it converges at a certain point.

Most of the reported figures regarding the impact of different factors on the performance were as expected. However, repeating the experiment setup of the baseline led to very different results. DLSI on 4 nodes (8 logical workers) took more than 8 hours which is not comparable to the 1h 41m reported in [49]. The distributed file system of RAAD could be one reason for that, as the baseline experiment reported in [30] and [49] was conducted on normal file systems. Another reason could be network latency between the nodes of RAAD, they may be communicating with multiple nodes for different jobs at the same time. In the baseline local machines were connected via Ethernet and exclusively used for DLSI.

5.2. Part 2: LSI on Encrypted Data

5.2.1. Baseline

The table below shows the results of LSI in the baseline solution. The query used is “Human Computer Interaction” as mentioned in section 1.4, where all document coded with c letter are related to the topic of human computer interaction.

Table 5: Documents Ranking Applying Baseline Setup

DOCID	CODES	DOCUMENTS	BASELINE
4	c5	Relation of user perceived response time to error measurement	0.21277076
1	c2	A survey of user opinion of computer system response time	0.212152261
2	c3	The EPS user interface management system	0.187983282
0	c1	Human machine interface for lab abc computer applications	0.187382123
3	c4	System and human system engineering testing of EPS	0.176122014
8	m4	Graph minors A survey	0.098594039
7	m3	Graph minors IV Widths of trees and well quasi ordering	0.069507883
6	m2	The intersection graph of paths in trees	0.06796983
5	m1	The generation of random binary unordered trees	0.064351132

5.2.2. Experiment 1

The table below shows how experiment 1 fails to hide the matrix A as the c-coded documents were not ranked at the top. The results in the table below are not similar to the baseline.

Table 6: Document Ranked According to their Normalized Similarities in Experiment 1

DOCID	CODES	DOCUMENTS	SIMILARITIES
4	c5	Relation of user perceived response time to error measurement	0.911735745
8	m4	Graph minors A survey	0.474022533
6	m2	The intersection graph of paths in trees	-0.578188074
5	m1	The generation of random binary unordered trees	-0.627143915
1	c2	A survey of user opinion of computer system response time	-0.651321413
2	c3	The EPS user interface management system	-0.671500085
7	m3	Graph minors IV Widths of trees and well quasi ordering	-0.965122498
0	c1	Human machine interface for lab abc computer applications	-1.351588316
3	c4	System and human system engineering testing of EPS	-2.676347817

5.2.3. Experiment 2

The aim of experiments 2 was to hide the matrix A as well. However, it applies a random matrix on the results of the matrix factorization rather than the matrix itself (see section 4.2.3.4). The table below shows how it fails to achieve acceptable effectiveness score as three of the c-coded documents are retrieved last.

Table 7: Document Ranked According to their Normalized Similarities in Experiment 2

DOCID	CODES	DOCUMENTS	SIMILARITIES
1	c2	A survey of user opinion of computer system response time	0.525876314
6	m2	The intersection graph of paths in trees	0.508173366
7	m3	Graph minors IV Widths of trees and well quasi ordering	0.493835488
4	c5	Relation of user perceived response time to error measurement	0.463864958
8	m4	Graph minors A survey	0.381556386
5	m1	The generation of random binary unordered trees	0.10405354
0	c1	Human machine interface for lab abc computer applications	0.042477934
2	c3	The EPS user interface management system	0.031139106
3	c4	System and human system engineering testing of EPS	-0.019685741

5.2.4. Experiment 3

Experiment 3 shifted its goal to hide A_k as formerly mentioned. The results shown in the table below indicates that it managed to achieve that goal it achieved the same results as the baseline (justification and mathematical prove are provided in section 5.2.7).

Table 8: Documents Ranking in Experiment 3

DOCID	**CODES**	**DOCUMENTS**	**A_k SPLITTED**	**BASELINE**
4	c5	Relation of user perceived response time to error measurement	0.212771	0.212771
1	c2	A survey of user opinion of computer system response time	0.212152	0.212152
2	c3	The EPS user interface management system	0.187983	0.187983
0	c1	Human machine interface for lab abc computer applications	0.187382	0.187382
3	c4	System and human system engineering testing of EPS	0.176122	0.176122
8	m4	Graph minors A survey	0.098594	0.098594
7	m3	Graph minors IV Widths of trees and well quasi ordering	0.069508	0.069508
6	m2	The intersection graph of paths in trees	0.06797	0.06797
5	m1	The generation of random binary unordered trees	0.064351	0.064351

5.2.5. Experiment 4

Experiment 4's goal was to hide the query from the cloud server. Details later discussed show how q_k was hidden using a random matrix r. The results shown in the table below indicates that similar to experiment 3 the experiment managed to achieve the same results as the baseline.

Table 9: Documents Ranking in Experiment 4

DOCID	CODES	DOCUMENTS	SPLITTED	BASELINE
4	c5	Relation of user perceived response time to error measurement	0.212771	0.212771
1	c2	A survey of user opinion of computer system response time	0.212152	0.212152
2	c3	The EPS user interface management system	0.187983	0.187983
0	c1	Human machine interface for lab abc computer applications	0.187382	0.187382
3	c4	System and human system engineering testing of EPS	0.176122	0.176122
8	m4	Graph minors A survey	0.098594	0.098594
7	m3	Graph minors IV Widths of trees and well quasi ordering	0.069508	0.069508
6	m2	The intersection graph of paths in trees	0.06797	0.06797
5	m1	The generation of random binary unordered trees	0.064351	0.064351

5.2.6. Experiment 5

Experiment five combines techniques of the previous two experiments, and similar to both of them it manages to achieve the same results as the baseline.

Table 10: Documents Ranking in Experiment 5

DOCID	CODES	DOCUMENTS	SPLITTED	BASELINE
4	c5	Relation of user perceived response time to error measurement	0.21277076	0.21277076
1	c2	A survey of user opinion of computer system response time	0.212152261	0.212152261
2	c3	The EPS user interface management system	0.187983282	0.187983282
0	c1	Human machine interface for lab abc computer applications	0.187382123	0.187382123
3	c4	System and human system engineering testing of EPS	0.176122014	0.176122014
8	m4	Graph minors A survey	0.098594039	0.098594039
7	m3	Graph minors IV Widths of trees and well quasi ordering	0.069507883	0.069507883
6	m2	The intersection graph of paths in trees	0.06796983	0.06796983
5	m1	The generation of random binary unordered trees	0.064351132	0.064351132

Experiment 5 on Medline

The figure below shows the effectiveness of LSI in experiment 5 compared to the baseline. Similar to experiment 3,4 and 5, the AP figures were exact in all 30 queries meaning the MAP is the same as shown by the horizontal line.

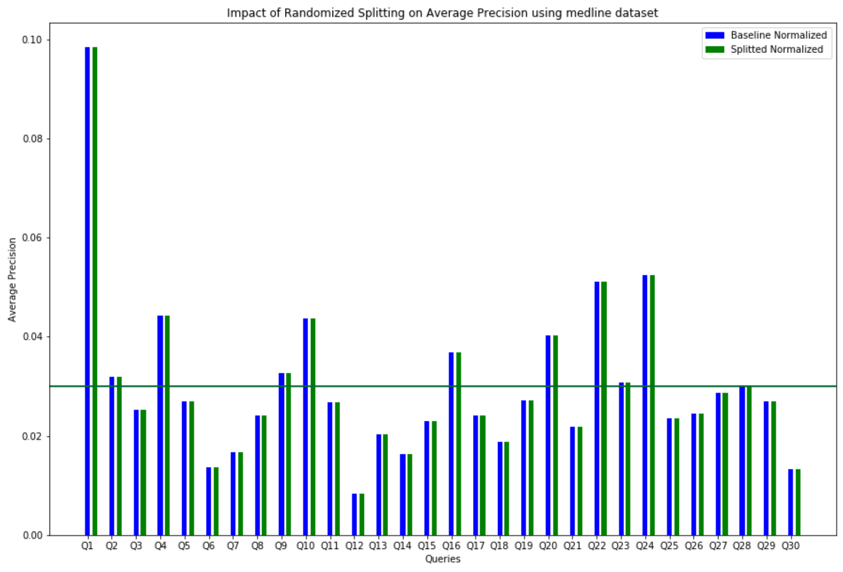


Figure 16: Impact of Randomized Splitting on Average Precision in Medline dataset

5.2.7. Discussion

In this set of experiments, it was shown that applying LSI on encrypted data is possible. First, experiment 3 showed that it is possible to hide the index/approximated matrix A_k from the cloud server. Second, experiment 4 showed that it is possible to hide the query from the cloud server. Finally, experiment 5 combined the work of both experiments, showing that it is possible to hide the index and the query from cloud server at the same time.

In all experiments, the similarities were exact to the baseline. The reason our proposed protocol works can be backed by a simple mathematical proof as shown below.

If we assume that both A_k and q_k are normalized first before calculating the cosine similarity. Then $\cos(A_k, q_k)$ will equal $q_k * A_k$ (note: summation was ignored assuming just one term for simplicity).

Now given that

$$A_{1k} = A_k + R \text{ , } A_{2k} = -R$$

$$q_{1k} = q_k + r \text{ , } q_{2k} = -r$$

Thus $\cos(q_{1k}, A_{1k}) + \cos(q_{2k}, A_{1k}) + \cos(q_{1k}, A_{2k}) + \cos(q_{2k}, A_{2k})$ will equal

$= (A_k + R) * (q_k + r) + (A_k + R) * (-r) + (-R) * (q_k + r) + (-R) * (-r)$

$= (A_k + R) * (q_k + r - r) + (-R) * (q_k + r - r)$

$= (A_k + R) * q_k + (-R) * q_k$

$= q_k (A_k + R - R) = q_k * A_k$

Therefore, random splitting should not yield the exact same result as the baseline and thus keep the effectiveness of the system intact.

In our experiments, we adopt random splitting technique since it is homomorphic with respect to LSI. Our encryption scheme needs homomorphic encryption to maintain the results of the cosine similarity and additions. This means that cosine similarity of the original query q_k and the matrix A_k (i.e. $\cos(q_k, A_k)$ or cosine of the additions) should be equal to the addition of the cosine similarities of q_{1k}, q_{2k}, A_{1k} and A_{2k} (i.e.

$\cos(q_{1k}, A_{1k}) + \cos(q_{2k}, A_{1k}) + \cos(q_{1k}, A_{2k}) + \cos(q_{2k}, A_{2k}))$. Maintaining homomorphic property in other techniques like the private/public key encryption has been the focus of some research papers after 2010 like [60] and [61], however efficiently applying homomorphic public to private key encryption is still a research area.

Chapter 6: Conclusion and Future Work

6.1.Overview

In this thesis two main goals were set and reflected into two main parts; part 1 investigates the scalability of the LSI implementation provided by gensim. Part 2 proposed a system to securely search over encrypted data that is indexed using LSI using free-text queries. The problem was formally defined describing the LSI's challenges and the need for scalable LSI solution in addition to the motivation behind applying LSI on encrypted data. LSI variations were reviewed including the work behind the gensim framework, Subspace tracking for LSI along with application of LSI in the literature. In addition, RAAD was introduced as a distributed environment leverage to applying gensim's distributed LSI. The conducted experiments are then reported along with discussions of insights and limitations. Several methods for applying LSI on encrypted data were proposed as well. The validity and results of those variation were then reported.

6.2.Achievements

The experiments conducted in part 1 show that the gensim LSI's is scalable, where DLSI performance is proportional to the number of workers used in the cluster. In general, DLSI outperformed serial LSI, considering some environment. The experiments also show that the effectiveness of LSI is not affected when distributed mode is used. The experiments have also raised a few limitations reported in the following section.

Part 2 shows that the proposed system managed to hide both the index/LSI model A_k and query q from the cloud server while maintaining the search effectiveness. Results show that the effectiveness was not at all affected as the similarities were always exactly as the baseline.

6.3.Limitations

On the LSI performance part, few challenges have risen while using gensim which limit the performance based on the user’s resources. First challenge was the memory limit; while gensim does not need to store the whole corpus in the memory as it is memory independent, yet there seems to be a minimal requirement. For example, distributed LSI on the 4M repeated corpus using 8 workers required a minimum 16 GB memory. Second, is the number of topics; gensim uses Pyro to communicate serialized objects between the cluster entities. However, since Pyro limits the size of serialized object to 2 GB this can raise some issues. For example, Pyro raised an error when applying DLSI on the 100k Wikipedia dataset and using 400 as the number of topic. Third, is the dictionary size, a dataset with narrow collection of topics (and in turn vocabs) is likely to be processed faster than one that is highly diverse. Last, the gensim library does not provide an interface to assign the host and port of the nameserver. While it is assumed that other entities will automatically locate the nameserver, we faced some issues initializing it and had to manually alter the library.

As for the security part, generating the topic model (i.e. LSI model in this work) is assumed to be on the client side. While this is no different from what is introduced in the literature, it is still a computationally intensive task.

6.4.Future Work

The motivation behind gensim library is to provide a scalable solution for some of the existing algorithms like LSI. Since it is an open source library, a potential future work is to alternate the library to support search over encrypted data which mitigates the challenges of scalability and memory management. This combines the output of both parts of this work; LSI scalability and LSI over encrypted data.

Another potential future work to integrate with, is the work of [25]. As discussed in section 3.1.4.2, [25] provides a methodology to scalably calculate document similarity with existing documents representation (Phase 3 in Figure 4) while genism provides a framework to scalably calculate those document representations (Phase 2 in Figure 4). Integrating both systems is possible noting that genism is an open source library.

REFERENCES

[1] C.D. Manning, P. Ragahvan, H. Schutze, An Introduction to Information Retrieval, Inf. Retr. Boston. (2009) 1–18. doi:10.1109/LPT.2009.2020494.

[2] S.T. Dumais, G.W. Furnas, T.K. Landauer, S. Deerwester, R. Harshman, Using latent semantic analysis to improve access to textual information, Proc. SIGCHI Conf. Hum. Factors Comput. Syst. - CHI ’88. (1988) 281–285. doi:10.1145/57167.57214.

[3] S.T. Dumais, Latent semantic analysis, Annu. Rev. Inf. Sci. Technol. 3 (2008) 4356. doi:10.4249/scholarpedia.4356.

[4] S. Deerwester, S.T. Dumais, R. Harshman, Indexing by latent semantic analysis, J. Am. Soc. Inf. Sci. 41 (1990) 391–407. doi:10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

[5] D.R. Tobergte, S. Curtis, Indexing by Latent Semantic Analysis, J. Chem. Inf. Model. 53 (2013) 1689–1699. doi:10.1017/CBO9781107415324.004.

[6] Q.N.R. Fund, QATAR NATIONAL RESEARCH STRATEGY (QNRS), (n.d.). https://www.qnrf.org/en-us/About-Us/QNRS.

[7] Qatar National Research Fund, Qatar National Research Strategy (QNRS) 2012 and a Strategic Plan for Implementation - Summary Version, (2012).

[8] Qatar Foundation, Thematic Pillars of Research Qatar ’ s Cross-cutting Research Grand Challenges, (n.d.) 1–8.

[9] Qatar Foundation, Qatar National Research Strategy 2014, (2014) 1–26. http://www.qnrf.org/Portals/0/Download/QNRS 2014.pdf.

[10] C. Bösch, P. Hartel, W. Jonker, A. Peter, A Survey of Provably Secure Searchable Encryption, ACM Comput. Surv. 47 (2014) 1–51. doi:10.1145/2636328.

[11] F. Baldimtsi, O. Ohrimenko, Sorting and searching behind the curtain, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 8975 (2015) 127–146. doi:10.1007/978-3-662-47854-7_8.

[12] T. Moataz, A. Shikfa, Boolean symmetric searchable encryption, Proc. 8th ACM SIGSAC Symp. Information, Comput. Commun. Secur. - ASIA CCS '13. (2013) 265. doi:10.1145/2484313.2484347.

[13] D. Boneh, B. Waters, Conjunctive, Subset, and Range Queries on Encrypted Data, TCC 2007 Theory Cryptogr. 4392 (2007) 535–554. doi:10.1007/978-3-540-70936-7.

[14] T.H.C.D.S. E. Shi J. Bethencourt, A. Perrig, Multi-dimensional range query over encrypted data, IEEE Symp. Secur. Priv. (2007) 350–364.

[15] J. Katz, A. Sahai, B. Waters, Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products, Adv. Cryptol. - EUROCRYPT 2008, 27th Annu. Int. Conf. Theory Appl. Cryptogr. Tech. Istanbul, Turkey, April 13-17, 2008. Proc. 4965 (2008) 146–162. doi:10.1007/978-3-540-78967-3_9.

[16] P. Sojka, Software Framework for Topic Modelling with Large Corpora, (2010).

[17] C. Wang, N. Cao, J. Li, K. Ren, W. Lou, Secure ranked keyword search over encrypted cloud data, Proc. - Int. Conf. Distrib. Comput. Syst. (2010) 253–262. doi:10.1109/ICDCS.2010.34.

[18] Q. Wang, J. Xu, H. Li, N. Craswell, Regularized latent semantic indexing, Proc.

34th Int. ACM SIGIR Conf. Res. Dev. Inf. - SIGIR ’11. (2011) 685. doi:10.1145/2009916.2010008.

[19] T. Hofmann, Probabilistic latent semantic indexing, Proc. 22nd Annu. Int. ACM SIGIR Conf. Res. Dev. Inf. Retr. (1999) 50–57. doi:10.1021/ac801303x.

[20] J. FEI, PLSI: A C++ implementation of probabilistic latent semantic indexing, (n.d.). https://github.com/fei3189/PLSI%0A (accessed December 31, 2017).

[21] S. Buss, cs_plsi: A CSparse PLSI implementation, (n.d.). https://github.com/sbuss/cs_plsi%0A (accessed December 31, 2017).

[22] Y. Chen, H. Zhang, Y. Zuo, D. Wang, An improved regularized latent semantic indexing with L1/2 regularization and non-negative constraints, Proc. - 16th IEEE Int. Conf. Comput. Sci. Eng. CSE 2013. (2013) 1075–1082. doi:10.1109/CSE.2013.156.

[23] Y.H. Zhicheng He, Yingjie Xu, Jun Xu, Maoqiang Xie, rlsi-java-source, (n.d.). https://code.google.com/archive/p/rlsi-java-source/ (accessed December 31, 2017).

[24] A. Mirzal, Similarity-based matrix completion algorithm for latent semantic indexing, Proc. - 2013 IEEE Int. Conf. Control Syst. Comput. Eng. ICCSCE 2013. (2013) 79–84. doi:10.1109/ICCSCE.2013.6719936.

[25] T. Elsayed, J. Lin, D.W. Oard, Pairwise document similarity in large collections with MapReduce, in: Proc. 46th Annu. Meet. Assoc. Comput. Linguist. Hum. Lang. Technol. Short Pap. - HLT ’08, 2008: p. 265. doi:10.3115/1557690.1557767.

[26] A. Atreya, C. Elkan, Latent semantic indexing (LSI) fails for TREC collections,

ACM SIGKDD Explor. Newsl. 12 (2011) 5. doi:10.1145/1964897.1964900.

[27] G. Gorrell, Generalized Hebbian Algorithm for Incremental Singular Value Decomposition in Natural Language Processing, Supercomputer. (2006) 97–104.

[28] M. Brand, Fast low-rank modifications of the thin singular value decomposition, Linear Algebra Appl. 415 (2006) 20–30. doi:10.1016/j.laa.2005.07.021.

[29] R. Řehůřek, Subspace Tracking for Latent Semantic Analysis, ECIR'11 Proc. 33rd Eur. Conf. Adv. Inf. Retr. (2011) 289–300. doi:10.1007/978-3-642-20161-5_29.

[30] R. Reh Ru, Fast and Faster: A Comparison of Two Streamed Matrix Decomposition Algorithms, NIPS 2010 Work. Lowrank Methods Largescale Mach. Learn. 6 (2011) 1–7. http://arxiv.org/abs/1102.5597.

[31] N.N. Chan, W. Gaaloul, S. Tata, A Web Service Recommender System Using Vector Space Model and Latent Semantic Indexing, (2011) 602–609. doi:10.1109/AINA.2011.99.

[32] Y. Kim, Y. Park, K. Shim, DIGTOBI : A Recommendation System for Digg Articles using Probabilistic Modeling, Www. (2013) 691–701.

[33] H. Park, K. Kwon, A.Z. Khiati, J. Lee, I.-J. Chung, Agglomerative Hierarchical Clustering for Information Retrieval Using Latent Semantic Index, 2015 IEEE Int. Conf. Smart City/SocialCom/SustainCom. (2015) 426–431. doi:10.1109/SmartCity.2015.108.

[34] O. El Midaoui, M. V-agdal, A. El Qadi, M. V-agdal, M. V-agdal, A New Approach to build a Geographical Taxonomy Semantic Indexing Method, (2015).

[35] H. Shahriar, V. Clincy, Anomalous Android Application Detection with Latent

Semantic Indexing, 2016 IEEE 40th Annu. Comput. Softw. Appl. Conf. (2016) 624–625. doi:10.1109/COMPSAC.2016.3.

[36] V. Bauer, T. Volke, S. Eder, Combining clone detection and latent semantic indexing to detect re-implementations, 2016 IEEE 23rd Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2016. (2016) 23–29. doi:10.1109/SANER.2016.26.

[37] M. Alsallal, R. Iqbal, S. Amin, A. James, Intrinsic Plagiarism Detection Using Latent Semantic Indexing and Stylometry, Dev. eSystems Eng. (DeSE), 2013 Sixth Int. Conf. (2013) 145–150. doi:10.1109/DeSE.2013.34.

[38] S. Hao, Y. Xu, H. Peng, K. Su, D. Ke, Automated chinese essay scoring from topic perspective using regularized latent semantic indexing, Proc. - Int. Conf. Pattern Recognit. (2014) 3092–3097. doi:10.1109/ICPR.2014.533.

[39] I. Toure, A. Gangopadhyay, Analyzing terror attacks using latent semantic indexing, 2013 IEEE Int. Symp. Technol. Homel. Secur. (2013) 334–337. doi:10.1109/THS.2013.6699024.

[40] A. Kontostathis, K. Reynolds, A. Garron, L. Edwards, Detecting cyberbullying: Query terms and techniques, Proc. 3rd Annu. ACM Web Sci. Conf. WebSci 2013. (2013) 195–204. doi:10.1145/2464464.2464499.

[41] J.L. Bigelow, A. Edwards, L. Edwards, Detecting Cyberbullying using Latent Semantic Indexing, (2016) 11–14.

[42] R.B. Bradford, Use of latent semantic indexing to identify name variants in large data collections, IEEE ISI 2013 - 2013 IEEE Int. Conf. Intell. Secur. Informatics Big Data, Emergent Threat. Decis. Secur. Informatics. (2013) 27–32.

doi:10.1109/ISI.2013.6578781.

[43] C. Hsieh, S. Cheng, C. Chang, C. Lin, Automatic Liver Segmentation from CT Images Using Latent Semantic Indexing, i (n.d.) 2–7.

[44] B. Done, P. Khatri, A. Done, S. Draghici, Predicting Novel Human Gene Ontology Annotations Using Semantic Analysis, Ieee-Acm Trans. Comput. Biol. Bioinforma. 7 (2010) 91–99. doi:10.1109/TCBB.2008.29.

[45] K.S. Thakare, R. Manthalkar, A.M. Rajurkar, D. Deshapande, Video retrieval using singilar value decomposition and latent semantic indexing, (2012) 1–5. doi:10.1109/ICCICT.2012.6398229.

[46] J. Xiao, W. Zhou, Q. Tian, Exploring tag relevance for image tag re-ranking, Proc. 35th Int. ACM SIGIR Conf. Res. Dev. Inf. Retr. - SIGIR '12. (2012) 1069. doi:10.1145/2348283.2348473.

[47] Anaconda, :: Anaconda Cloud, (n.d.). https://anaconda.org/.

[48] Project Jupyter, Project Jupyter | Home, (n.d.). http://jupyter.org/.

[49] Radim Řehůřek, gensim: Distributed Latent Semantic Analysis, (n.d.). https://radimrehurek.com/gensim/dist_lsi.html.

[50] Radim Řehůřek, gensim: Introduction, (n.d.). https://radimrehurek.com/gensim/intro.html (accessed January 1, 2017).

[51] Texas A&M University at Qatar (TAMUQ), Research Computing > History of our High Performance Computing, (n.d.). https://rc.qatar.tamu.edu/Pages/HPC-History.aspx (accessed January 1, 2017).

[52] Texas A&M University at Qatar (TAMUQ), Raad-II Technical Summary, (n.d.).

https://rc.qatar.tamu.edu/Pages/hpc/raad2/specs.aspx (accessed January 1, 2017).

[53] Glasgow IDOM - Medline collection, (n.d.). http://ir.dcs.gla.ac.uk/resources/test_collections/medl/.

[54] Radim Řehůřek, gensim: Experiments on the English Wikipedia, (n.d.). https://radimrehurek.com/gensim/wiki.html.

[55] Radim Řehůřek, gensim: corpora.dictionary – Construct word<->id mappings, (n.d.). https://radimrehurek.com/gensim/corpora/dictionary.html (accessed January 1, 2017).

[56] WikiMedia, enwiki dump progress on 20171103, (n.d.). https://dumps.wikimedia.org/enwiki/20171103/.

[57] Radim Řehůřek, gensim: similarities.docsim – Document similarity queries, (n.d.). https://radimrehurek.com/gensim/similarities/docsim.html.

[58] T. Elsayed, IR is an Experimental Science ! Questions About the Black Box, (2015) 1–21.

[59] J.M. Rubin, MIT School of Engineering | » Can a computer generate a truly random number?, (n.d.). https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/.

[60] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proc. 41st Annu. ACM Symp. Symp. Theory Comput. - STOC ’09, 2009: p. 169. doi:10.1145/1536414.1536440.

[61] N.P. Smart, F. Vercauteren, Fully homomorphic encryption with relatively small key and ciphertext sizes, in: Lect. Notes Comput. Sci. (Including Subser. Lect.

Notes Artif. Intell. Lect. Notes Bioinformatics), 2010: pp. 420–443.
doi:10.1007/978-3-642-13013-7_25.

APPENDIX I: LIST OF ALL RAAD EXPERIEMENTS

Table 11: CPU recorded time for LSI on 100 k documents

NO. DOCUMENT	**113550 (≈100K)**				
NO. MACHINES	No. Workers	Time			Time (s)
		h	m	s	
1 (SERIAL LSI)	N.A		31	39	1899
1	2		17	26	1046
2	4		15	29	929
3	6		14	14	854
4	8		13	55	835

Table 12: CPU recorded time for LSI on 500 k documents

NO. DOCUMENT	**500 K**				
NO. MACHINES	No. Workers	Time			Time (s)
		h	m	s	
1 (SERIAL LSI)	N.A	2	17	50	8270
1	2	1	12	45	4365
2	4		42	49	2569
3	6		34	0	2040
4	8		32	6	1926

Table 13: CPU recorded time for LSI on 1 M documents

NO. DOCUMENT	**1 M**				
NO. MACHINES	No. Workers	Time h	m	s	Time (s)
1 (SERIAL LSI)	N.A	4	59	57	17997
1	2	2	22	11	8531
2	4	1	17	14	4634
3	6		57	36	3456
4	8		50	13	3013

Table 14: CPU recorded time for LSI on 1.5 M documents

NO. DOCUMENT	**1.5 M**				
NO. MACHINES	No. Workers	Time			Time (s)
		h	m	s	
1 (SERIAL LSI)	N.A	7	7	29	25649
1	2	3	34	42	12882
2	4	1	51	3	6663
3	6	1	20	52	4852
4	8	1	7	13	4033

Table 15: CPU recorded time for LSI on 2 M documents

NO. DOCUMENT	**2 M**				
NO. MACHINES	No. Workers	Time			Time (s)
		h	m	s	
1 (SERIAL LSI)	N.A	9	27	33	34053
1	2	4	37	7	16627
2	4	2	25	13	8713
3	6	1	42	42	6162
4	8	1	24	16	5056