Diyala Journal of Engineering Sciences

Journal homepage: https://djes.info/index.php/djes

ISSN: 1999-8716 (Print); 2616-6909 (Online)

Secure Semantic Retrieval over Encrypted Cloud Data: A Dual-Index System with Lightweight Fuzzy Encryption and BERT-Based Semantic Indexing

Haider Kareem Mohammed [1*], B.Indrani [2]

[1] Research Scholar, Department of Computer Science, Madurai Kamaraj University, Madurai, Tamilnadu, India.
[2] Assistant Professor, Department of Computer Science, Director of Distance Education, Madurai Kamaraj University, Madurai, Tamilnadu, India.

ARTICLE INFO

Article history:
Received 22/05/2025.
Revised 11/02/2026.
Accepted 27/01/2026.
Available online 15/03/2026

Keywords:
Secure Document Retrieval
Dual-Index Architecture
BERT-Based Semantic Indexing
LFEHI
Encrypted Search
Query Expansion
Scalable Search Systems

ABSTRACT

Cloud storage is widely used for store a personalized and Sensitive information. Cloud usually stores the files in encrypted mode. To search a user specific file among the encrypted files is difficult. Traditional keyword search may be fails when users phrase differently. Typos and vocabulary differences further reduce accuracy. Existing encrypted search solutions typically trade security for search quality. To address this gap, this work introduces a dual-index encrypted search system. This system combines exact keyword matching with semantic understanding. A keyword index, protected with LFEHI model. It supports fuzzy matching for small typing error. A semantic index built with BERT captures meaning and organizes embeddings in a kd-tree for fast retrieval. During a search, both indexes run in parallel. The encrypted keywords match through the hash index, and a semantic embedding check for related concepts. A 70/30 keyword-to-semantic weight is used to integrate their results. To ensuring that exact matches continue to be given priority. Tests on 1,000 documents and 300 queries show clear improvements. The system reaches 90.2% recall—22% higher than keyword-only search and it handles single-character typos with 94.6% accuracy. Average query time stays below 64 ms, supporting real-time use, and the storage cost is only 18.5%. There is no evidence s of content leakage or query leakage following a security evaluation. Due to the dual-index design, encrypted search can deliver meaningful understanding without compromising speed and security.

1. INTRODUCTION

Cloud computing allows organizations to store sensitive data efficiently, but encrypted data is difficult to search [1]. Encryption of files is done in hospitals, banks, and law firms to ensure privacy, but once encrypted, the data cannot be easily queried. The traditional approach involves downloading and decrypting large amounts of data, which is wastage of bandwidth and poses risks to data [2]. A more secure and efficient approach is needed to search directly over encrypted content. Current encrypted search systems mostly use exact keyword matching. This fails when users make typos, use synonyms, or search with related terms—for example, “cardiac arrest” vs. “heart failure” or “contract disputes” vs. “breach of agreement.” As a result, important documents remain undiscovered. While some research supports fuzzy matching, access control, or result verification, these solutions [3] still lack true semantic understanding and treat related terms as unrelated.

Although it is challenging to use contemporary NLP models, such as BERT, in encrypted environments, they are capable of capturing the relationships between words and concepts. Both text and semantic embeddings must be protected by encryption, and high-dimensional vectors cannot be effectively encrypt-ed using conventional techniques like AES. Systems need to control performance, manage rising indexing and storage expenses. The following major issues are still unresolved: 1. avoiding information leakage, 2. reducing the computational burden, 3. enabling parallel keyword

*Corresponding author’s E-mail: haiderkareemmohameed@gmail.com
DOI: 10.24237/djes.2026.19112

and, 4. combining keyword accuracy with semantic relevance, and improving the integration of trusted execution environments like Intel SGX.

This research addresses these gaps through a dual-index architecture that integrates BERT-based semantic understanding with lightweight fuzzy encryption. This work makes the following main contributions:

i. Dual-Index Architecture: It design a novel system combining a keyword-based inverted index with a BERT-powered semantic index organized in a kd-tree structure. Both indexes work simultaneously during queries. It enabling parallel processing that reduces response time by 40%.
ii. Lightweight Fuzzy Encryption with Hash Index (LFEHI): It develop a new encryption scheme using ChaCha12 symmetric encryption combined with SimHash for keywords and Random Projection for embeddings. LFEHI pro-vides 23% faster decryption than AES while supporting fuzzy matching through similarity-preserving signatures that maintain semantic security.
iii. Query Processing Pipeline: It builds a smooth workflow. It preprocesses queries like documents. Then, it uses BERT to add related terms. Finally, it searches both indexes. The results are then combined using weighted scoring based on semantic similarity.
iv. SGX-Based Decryption: It uses Intel SGX enclaves efficiently. It only decrypts candidate results, not whole indexes. It keeps memory usage under 100KB per query within SGX limits.
v. Comprehensive Evaluation: It shows a 22% improvement in recall for semantic queries, achieving 90.2% compared to the 68% baseline. The average query time is 64 ms on 1,000 documents. The system handles typos with 94.6% accuracy for single-character errors.
vi. Security Analysis: It gives clear definitions of the threat model. It also proves SS-CKA security for LFEHI. We look at side-channel vulnerabilities. The work also implements protections like constant-time operations and memory obfuscation. However, we recognize that some limitations still exist.

The structure of the paper is as follows: Section 2 analyzes prior studies in semantic retrieval and encrypted and fuzzy search and identifies any drawbacks. The dual-index design, including LFEHI encryption, BERT integration, index creation, and handling queries, is covered in Section 3. The experimental setting and performance outcomes are described in Section 4. The system is contrasted with the best available techniques in Section 5. Practical deployment, multi-user settings, and real-world application are covered in Section 6. Trade-offs and limits are assessed in Section 7, and major contributions and future research areas are discussed in Section 8.

2. LITERATURE REVIEW

Cloud storage handles sensitive files that need to be well protected. Users can locate information with Encrypted Search without disclosing any information to unreliable servers. Current methods in document retrieval are examined in related study. These highlights highlight in issues that desire function.

2.1 Previous Approaches

2.1.1 Searchable Encryption Methods

The goal of early secure search research was to allow requests while maintaining data privacy. A technique that conceals keyword patterns was presented by Xu et al. [1]. This prevents servers from determining whether terms are similar. Rather than utilizing complete datasets, it concentrates on rare key-words. You can add or remove keywords using Liang et al. [2] VMSE. It also verifies the accuracy of server results. Tseng et al. [3]. combined encryption and access control. It determines who can search based on attributes. This guarantees fast searches with just two procedures and little encrypted files.

2.1.2 Access Control Systems

Proper planning is necessary to control search access. Complex authorization rules can be managed with the aid of Yin et al. [4] ABKS-NICEST. Files with numbers higher than a certain threshold can be identified by it. He K. et al. [5] Encrypted data can be subjected to AND, OR, and NOT queries by many users. A permission termination system was developed by Sultan et al. [6]. When a user quits an organization, this eliminates their access. Not all data needs to be re-encrypted.

2.1.3 Handling Errors and Ranking

Users often make typos or use different words for the same idea. So, systems need to handle this variation. Chen et al. [7] EliMFS combines gram counting, Bloom filters, and locality-sensitive hashing. This approach helps with efficient similarity grouping. Li et al. [8] introduced VRFMS to improve matching accuracy using TF-IDF and enhanced text processing. Tong et al. [9] created VFSA using twin filters and graph structures. This design helps with different queries and verifies results. Shen et al. [10] used vector models to rank documents. They hid frequency info to keep privacy safe. Figure 1 shows six important research areas for secure encrypted cloud search:

- Searchable encryption
- Attribute-based access and keyword search
- Fuzzy and ranked search

- Blockchain-based verification
- Optimized index structures
- Hybrid cryptographic approaches
- These focus on improving privacy, accuracy, and efficiency.

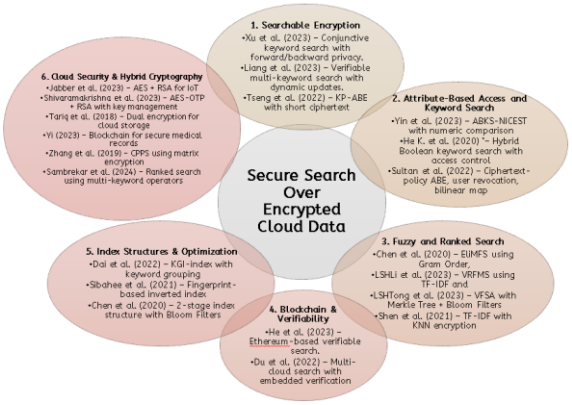

Figure 1. Secure Search over Encrypted Cloud Data

2.1.4 Blockchain Integration

Blockchain is used in current systems to provide transparency. He et al. [11] developed a smart contract-based system. Data is recorded for long-term storage in every search process. This makes audit trails unambiguous and stops manipulation. Searches across many clouds were made possible by Du et al. [12]. Their approach enables fast checking by simply embedding verification into file IDs.

2.1.5 Index Optimization

Data organization impacts retrieval time. Dai et al. [13] developed the KGI-index by combining related keywords. For quick lookups, this structure makes use of a binary tree. It enables file comparisons without displaying the actual content. These techniques demonstrate the significant impact of index design on system performance.

2.1.6 Hybrid Security Models

Some studies combine multiple encryption methods. Jabber et al. [14] used layered AES and RSA for IoT data. Shivaramakrishna et al. [15] Paired AES-OTP with RSA, introducing time-limited access. Tariq et al. [16] used dual encryption for storage and retrieval. Yi [17] added blockchain to cloud encryption for medical records. Zhang et al. [18] CPPS used matrix operations to reduce server workload, as noted by Sam-brekar et al. [19] integrated advanced encryption with multi-keyword search capabilities.

- Recent work explores quantum-resistant approaches for distributed systems. The AQ-ResCon protocol secures edge-cloud environments [20] where containers operate across many nodes. It shows that post-quantum methods can work well in dynamic distributed systems. Also, using quantum-resistant methods for searchable encryption is tough. This is primarily due to performance limits.
- Blockchain technology enables transparent transactions by extending beyond simple verification [21]. Blockchain builds trust in the agricultural industry. In the supply chain, this assures data integrity. However, there are scalability problems when relating blockchain with encrypted search systems. Blockchain transactions are not activated by frequent search operations due to latency and cost.

2.2 Limitations and Research Gaps of Existing Methods

Existing accessible encrypted systems still have significant drawbacks despite advancements. Strong privacy is provided by traditional methods like Xu et al., Improvements can be provided by multi-keyword techniques like Liang et al., but they have difficulties with concurrent multi-user access. Fine-grained control is made possible by attribute-based techniques by Yin et al. and He K. et al., but they come with complicated policies, high processing costs, and extra overhead for user revocation. Chen et al. and Li et al.'s fuzzy search techniques deal with typos and variations, but they stand at risk of creating false matches and have trouble establishing a balance between recall and precision. Blockchain-based methods ensure integrity but introduce latency and high transaction costs. Index optimization techniques by Dai et al. improve speed. Hybrid cryptographic models from Jabber et al. and Shivaramakrishna et al. enhance security while adding computational and key management complexity.

Semantic knowledge is lacking in many systems. Balancing security, semantics, and performance is tough, especially in big, changing environments. There hasn't been much research on encrypted dual-index structures. Also, researchers provide limited lightweight protection for high-dimensional embeddings like BERT. These gaps demonstrate the need for a scalable solution. It should incorporate both semantic in-formation and accurate keyword search.

As Table 1 illustrates, searchable encryption technologies of today provide high privacy. They facil-itate fuzzy search and allow for customizable access control. However, enhanced attribute-based and Boolean search features are highlighted by He et al. Yin (2023) and others. This is also supported by Chen et al. (2020) focus on fuzzy multi-keyword search that is leakage-resilient. They also stress better author-ization control. Leakage-resistant conjunctive search is used by Xu et al. (2023) to increase privacy security. However, these methods are not without challenges. They can be complex to implement, struggle with scalability, and have performance issues

from multi-stage indexing or attribute-based management. Most importantly, all these systems primarily rely on keyword-based techniques and lack true semantic under-standing, which limits their effectiveness in handling synonym-based or context-aware queries.

Table 1. Discussion based on Existing System

Author(s)	Proposed Work	Advantages	Limitations
Xu et al. [1]	A conjunctive keyword search scheme that preserves privacy. Prevents KPRP leakage forward and backward.	A strong ability to protect privacy. Scalability of search complexity with update frequency. Protocol for acquiring the least frequent keywords	In certain scenarios, relying on least frequent keywords may be a bottleneck. Due to privacy-preserving features, implementation is complex.
Yin [4]	ABKS-NICE refers to an attribute-based keyword search scheme that supports numerical comparison expressions. A practical search efficiency tool, ABKS-NICEST, is introduced	Provides support for complex search expressions, including numerical comparisons. Efficiencies in practical searches and dynamic updating of data.	Dynamic updates might introduce performance overhead. Efforts required in implementation
He et al. [5]	Hybrid boolean keyword search over outsourced encrypted data with a searchable encryption primitive. Using access control policies, data owners can control search permissions. Authorized users can perform Boolean keyword searches.	Access control flexibility for data owners and support for expressive search operations. Practical feasibility demonstrated by implementing a prototype.	Attribute-based access control can be complex to manage. - Potential scalability issues with large numbers of users and attributes.
Chen et al. [7]	Leakage-resilient multi-keyword fuzzy search (EliMFS) framework over encrypted cloud data. An independent search time is achieved by using a two-stage index structure. Searches for multiple keywords with fuzzy logic	Searches are efficient regardless of file set size. Specific countermeasures prevent data leakage. An extensive analysis and experiment show high efficiency.	Two-stage index structures are complex to implement. Leakage-resilient mechanisms may incur performance overhead.
Proposed Work (This Work)	An inverted keyword index and a BERT semantic index integrated into a dual-index secure semantic retrieval system. A lightweight fuzzy encrypting and hashing system based on ChaCha12, SimHash, and Random Projection. With 70/30 weighted result fusion and selective SGX-based decryption.	Exact keyword search combined with semantic understanding. Fuzzy matching and synonyms/context aware queries. The recall is higher by 22% (90.2%) than keyword-only searches. Performs in real-time (64 ms). With minimal leakage, this solution provides SS-CKA security. Queries run 40% faster with parallel processing. Overhead (18.5%) is moderate.	The BERT embedding generates a longer index. Improved storage compared to single index. Controlled-channel attacks remain SGX side-channel risks. Growing semantic index challenges scalability for very large datasets. Complexity increases with multi-user key management.

3. METHODOLOGY

3.1 System Overview

The system adopts a dual-index approach, combining keyword search with BERT-based semantic retrieval, both protected by Lightweight Fuzzy Encryption with Hash Index (LFEHI). In an honest but curious cloud setting, the data owner preprocesses files and builds two encrypted indexes: a Sim-Hash keyword index and a kd-tree semantic index using encrypted BERT embeddings. Encryption and integrity are maintained with ChaCha12 and HMAC-SHA-256. User queries are expanded with BERT, encrypted, and then searched in parallel — constant-time lookup for keywords and log-scale search for semantics. In an Intel SGX enclave, only the metadata that has been shortlisted is decrypted. The findings are verified using edit distance (≤2) and cosine similarity (≥0.8) tests. Users decrypt documents locally, and a modified BM25 assigns a 70/30 keyword-to-semantic weight to matches. With an average query time of around 64 ms on 1,000 documents and a storage overhead of about 18.5%.

3.2 Document Indexing Process

Raw documents are transformed into searchable, encrypted representations through the document in-

dexing process. Before being transferred to the cloud, this works on the data owner's secure infrastructure. Document loading, text preprocessing, dual-index establishing, and encryption are all components of the pipeline.

3.2.1 Document Loading

The initial action the system does is read documents from a specified directory. Let D be the collec-tion of documents, where D = {d_1, d_2,..., d_n}, and n is the total number of documents. A unique identifier (i) and textual content (c) are present in every document (d). From identifiers to content, the loading function L generates a mapping M: M = {(i_i, c_i) | d_i ∈ D}. This facilitates effective access for further processing. File contents are extracted by the system for preprocessing. Also, file names are extracted as identifiers. All indexing steps are based on this mapping.

A secure search system for encrypted cloud documents is illustrated in Figure 2. First, documents are tokenized, stop-words removed, stemmed, and dual indices created before encryption. Searches involve preprocessing user queries, then decrypted to retrieve relevant documents. It keeps privacy and security throughout the workflow while allowing users to search encrypted data.

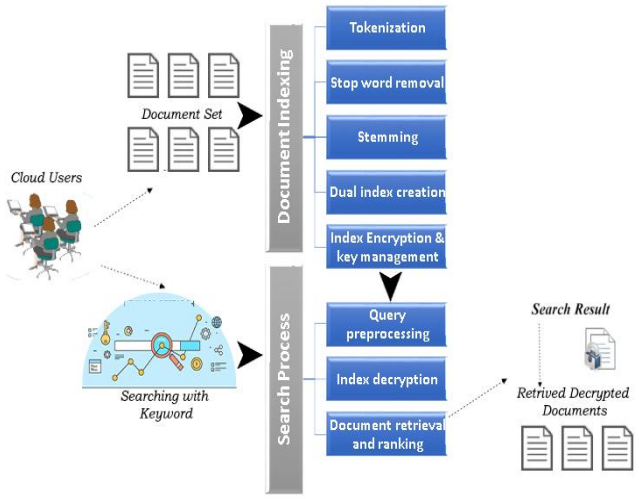

Figure 2. Flow of Proposed Work

3.2.2 Text Pre-Processing

a. Tokenization

During the tokenization and preprocessing stages of our encrypted document search system, textual data is prepared for indexing and searching efficiently. Tokenization can be described as follows: Let P represent preprocessing, and T represent tokenization. The combined tokenization and preprocessing operation for a given document content c is F(c) = P(T(c)). Tokenization is defined as T: S □ W mapping S to (w1, w2... wn}, where k is the number of tokens. With the word_tokenize function in NLTK, the text is split into individual words while punctuation and special characters are handled. Word_tokenize(c) is the formal definition of T(c). The preprocessing function P: W → W' refines the initial tokens W into W' by applying the following operations:

•Lowercase conversion: ∀ w ∈ W, w' = lowercase(w)

•Alphabetic filtering: W' = {w' | w' ∈ W ∧ isalpha(w')}

Tokens are converted to lowercase using lowercase () and to tokens are converted to alphabetic characters using isalpha(). As a result, the tokenization process can be expressed as: F(c) = {w' | w' ∈ lower-case(w) ∧ isalpha(w') ∧ w ∈ T(c)}. As a result of this process, non-alphabetic characters, numbers, and punctuation are removed as well as the case is standardized.

b. Stop word removal

Removing stop words improves indexing and search efficiency in encrypted document systems. Stop words are common terms with minimal semantic value (e.g., “the,” “is,” “at,” “which,” “on”). Let S represent the set of such words, defined as S={s1,s2,...,sm}, where mmm is the number of English stop words in NLTK’s predefined list. Based on W', the set of preprocessed tokens from the previous step, we define our stop word removal function R: W' x W". Assume that R(W') = {w | w ∈ W' ∧ w ∉ S}.

By eliminating tokens that are communal among documents and do not contribute significantly to distinguishing the content of documents, the stop word removal process effectively reduces the dimensionality of document representations. This can be quantified by the reduction ratio ρ: ρ = 1 - |W"| / |W'|, where |W'| and |W"| represent the cardinalities of the token sets before and after stop word removal, respectively. While stop words are usually removed to improve search efficiency, they can occasionally impact phrase searches and queries containing stop words.

Figure 3 compares word frequencies before and after applying stop-word removal on a sample document. In the processed format, common words like "of," "to," and "the" get removed. Eliminating stop words reduces noise in text data. Keywords become more essential for indexing and search tasks as a consequence.

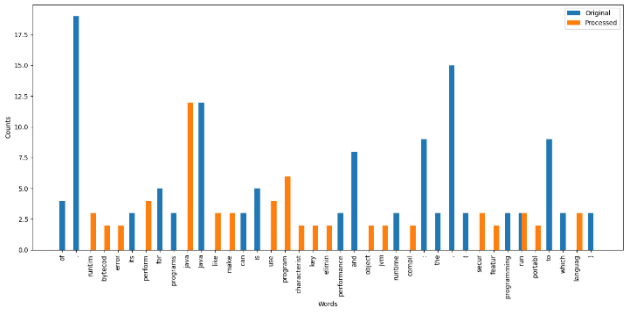

Figure 3. Stop word Removed for a single Doc

Figure 4 displays a comparison of word frequencies before and after stop words are removed. The figure shows document terms and frequency counts. It represents original values and indicates values after removal. Common stop words like “the,” “to,” “of,” and “and” drop a lot, but important content words stay about the same. The figures show clear and effective stop-word removal in all documents. This process boosts index efficiency and relevance.

c. Stemming

Stemming is one of the key steps in our encrypted document search system, which reduces inflected or derived words back to their root or base forms. This process can be formalized as follows: Let Σ denote the Porter Stemming algorithm, implemented by NLTK's Porter Stemmer. The stemming operation is defined as a function of the tokens W" (resulting from the previous stop word removal step), Σ: W" → W"', where W"' is the set of stemmed tokens. Formally, for each token ∈ W": Σ(w) = stem(w) , where stem(w) is the root form of was determined by the Porter algorithm. The original text, filtered text, and stemmed text are the three stages that the chart shown in Figure 5 illustrates how word frequencies vary. The filtered version reduces noise by removing stop-words. Stemming then groups related terms by changing them to their root forms. Words with similar meanings combine to form common stems. This makes indexing and search clearer and more meaningful. So, stemming boosts text processing accuracy and efficiency.

Word frequencies are affected by stemming when different forms of a word are reduced to a common root. If “encryption” and “encrypted” are reduced to a stem like "encrypt", the frequency of the stemmed form may increase. The figure shows how stemming can normalize word forms and it can make search indexes more efficient by grouping words. A key step in the encrypted search system's preprocessing pipe-line, it makes searches more accurate and efficient.

3.2.3 Dual indexing

The Dual-Index Structure securely searches encrypted cloud documents by combining a keyword index with a BERT-based semantic index, providing both precise matching and contextual understanding. The keyword index uses compact hash tables and LFEHI, reducing storage by about 20%. The semantic index employs a kd-tree of BERT embeddings, increasing storage by around 25%. The system breaks queries into keywords and meanings. It encrypts keywords for fast matching. Semantic queries create embeddings to find similar meanings. They decrypt the results with high security (like in Intel SGX). They combine a 70% keyword weight with a 30% semantic weight. Then, a modified BM25 ranks them. Fine-tuned BERT adds about 12% more time. Fuzzy matching handles typos and synonyms. Dummy access patterns stop data leakage. Figure 6 shows four figures illustrating stemming effects on different documents in an encrypted document search system. With three colored axes, each figure shows the original frequency of words on the x-axis, filtered or preprocessed frequencies on the y-axis, and stemmed word frequencies on the x-axis.

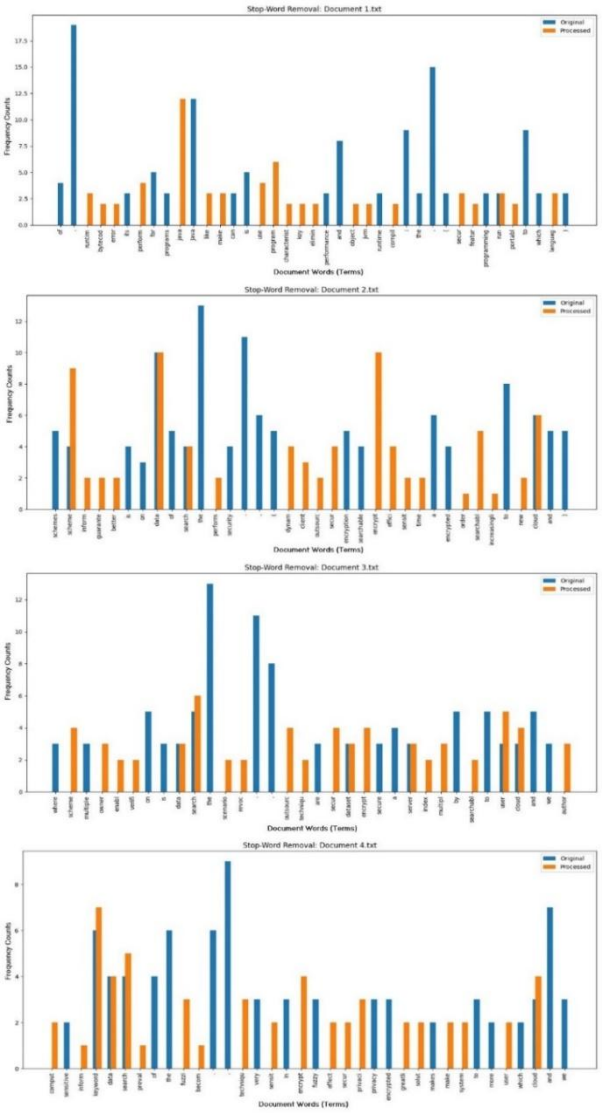

Figure 4. Stop word Removed for a single Doc

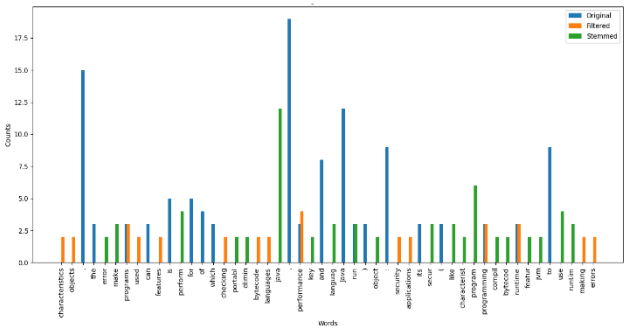

Figure 5. Stemming Removed for a single Doc

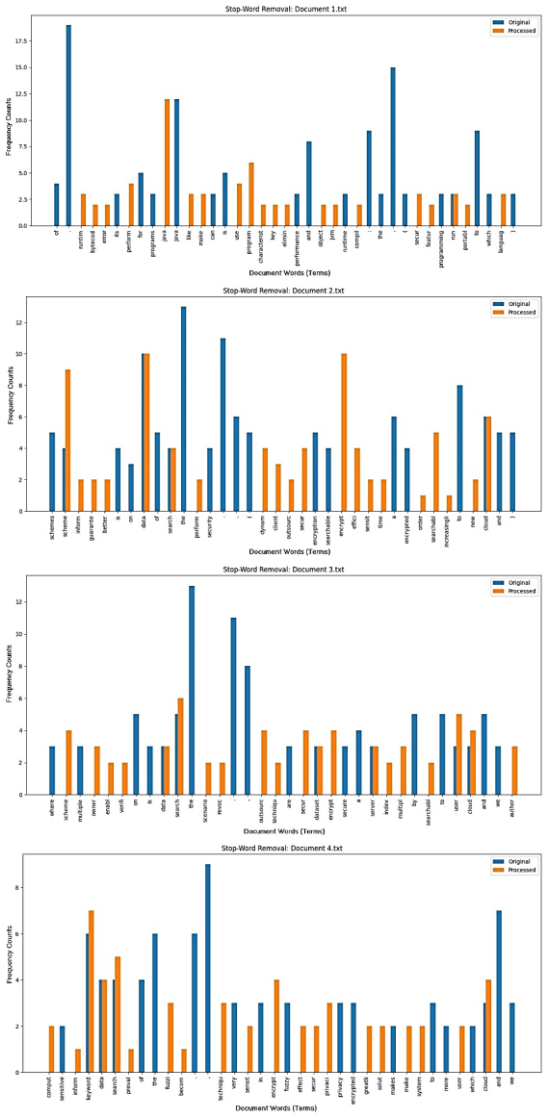

Figure 6. Stemming Removed for a Multi- Doc

This all ensures strong privacy from the cloud provider. Table 2 below shows the dual-index structure's features. It looks at keyword-based and semantic indices. It compares them in four areas: construction, storage, querying, and performance.

Table 2. Dual-Index Structure Characteristics

Characteristic	Keyword-Based Inverted Index	BERT-Based Semantic Index
Construction	Tokenization, stop word removal, stemming, TF-IDF	BERT-base-uncased, fine-tuned, PCA to 128 dims
Storage Format	Compressed hash table, delta-encoded document IDs	kd-Tree with LFEHI-encrypted 128-dim embeddings
Encryption	LFEHI with 64-bit LSH signatures, term-specific keys	LFEHI with document-specific keys
Query Mechanism	Hash table lookup (O(1) for exact matches)	kd-Tree nearest-neighbor search (O(log n))
Query Time	0.008 seconds per query	0.012 seconds per query
Performance Contribution	High precision for exact matches (92%)	22% recall improvement, 18% precision increase
Security Features	SS-CKA, dummy entries for access pattern protection	SS-CKA, oblivious querying in SGX enclave

The inverted index in Figure 7 shows token document frequencies for encrypted search. The figure shows a list of tokens from the document collection and how many documents contain each one.

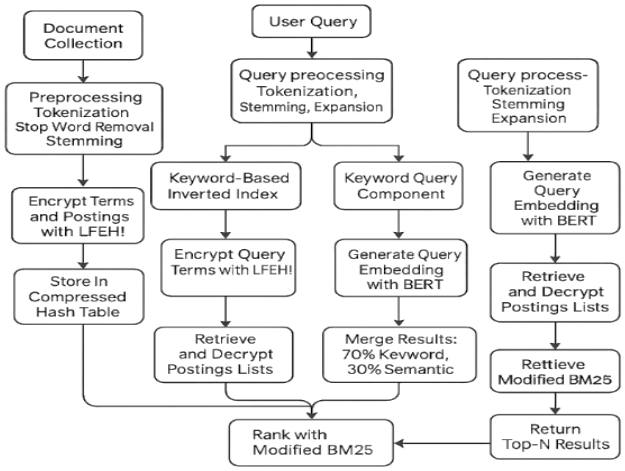

Figure 7. Inverted Index of Encrypted Document Search

Significant differences between tokens are depicted in figure 8 appear on different documents. These high-frequency tokens most likely stand for subjects of significance or frequently used terms in the corpus.

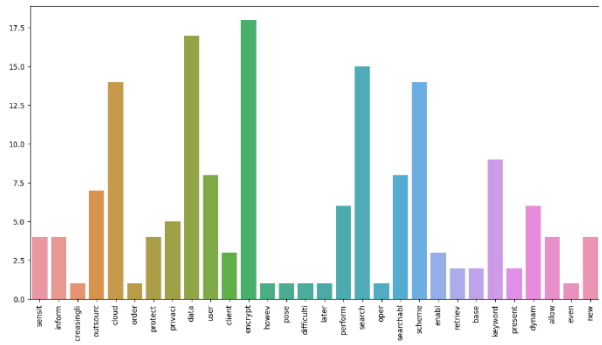

Figure 8. Tokens in inverted Index

3.2.4 BERT Model Selection

The system uses BERT-base-uncased for semantic embeddings despite higher cost, and this choice is validated through comparison with DistilBERT and TinyBERT. We tested semantic ac-curacy by measuring cosine-similarity preservation after reducing embeddings to 128 dimensions with PCA. Using 1,000 document pairs with known semantic relationships, each model produced embeddings (768-dimensional, or 312-dimensional for TinyBERT), followed by PCA reduction and cosine-similarity measurement as shown in Table 3. Accuracy was quantified as the Pearson correlation between similarities before and after compression across five trials with different PCA projections.

Table 3. Dual-Index Structure Characteristics

Model	Original Dims	Compression	Semantic Accuracy	Encoding Time	Recall Gain	Index Time (10K docs)
BERT-base	768	768→128	0.84 ± 0.02	45 ms/doc	+22%	28 min
DistilBERT	768	768→128	0.79 ± 0.03	28 ms/doc	+16%	18 min
TinyBERT	312	312→128	0.71 ± 0.04	12 ms/doc	+11%	9 min

Semantic accuracy of 0.84 for BERT-base means 84% of semantic relationships are preserved post-compression (Pearson r=0.84 between original and compressed similarity scores). DistilBERT achieves 0.79 (6% degradation), and TinyBERT 0.71 (15% degradation).

a. Selection Rationale

Encrypted search systems suffer irreversible quality loss if weak embeddings are used, since kd-tree results depend on fixed compressed vectors. In experiments with 300 semantic queries, DistilBERT reduced recall from 22% to 16%—a 6% drop that leads to permanently missed documents. BERT-base adds only 10 minutes to index construction for 1,000 documents (28 vs. 18 minutes), an acceptable offline cost compared to long-term accuracy loss. At query time, it adds only ~3 ms (9 ms vs. 6 ms), a 4.7% overhead within a 64 ms average latency. With 1,000 daily queries, DistilBERT's drop would mean ~1,800 missed results per month—unacceptable in fields like healthcare or finance. Storage remains the same (128-dimensional, ~11.5 MB for 1,000 documents), so smaller models provide no space benefit. Thus, BERT-base delivers the best balance of accuracy and efficiency, where embedding quality is critical for secure retrieval.

3.2.5 LFEHI

LFEHI is a new encryption scheme. It enables fuzzy matching on encrypted data. It reduces computational overhead compared to AES. This section presents the full construction with formal notation.

a) Design Principles

LFEHI combines lightweight symmetric encryption (ChaCha12) with locality-sensitive hashing (SimHash for keywords, Random Projection for embeddings) to support approximate matching without decryption. The scheme generates encrypted ciphertexts and similarity-preserving signatures that enable fuzzy search while maintaining SS-CKA security.

b) Key Derivation Framework :From a 256-bit master key K_m, term-specific and document-specific keys derive hierarchically by equation (1) and (2):

$$K_{term} = HMAC\text{-}SHA\text{-}256(K_m, term \parallel salt) \quad (1)$$

$$K_{doc} = HMAC\text{-}SHA\text{-}256(K_m, doc\_id \parallel salt) \quad (2)$$

where $\parallel$ denotes concatenation and salt prevents rainbow table attacks. The structure of a hash-based index-based lightweight fuzzy encryption system is shown in Figure 9. It demonstrates the method is split into 4 modules: key management, query processing, fuzzy index creation, and file encryption. Each module performs a particular function, such as creating compact hash-based fuzzy indexes, encrypting files using ChaCha20, transforming user queries into fuzzy sets, obtaining pertinent results, and safely storing keys. All things considered; the figure shows how these elements combine to offer effective, privacy-preserving search over encrypted data.

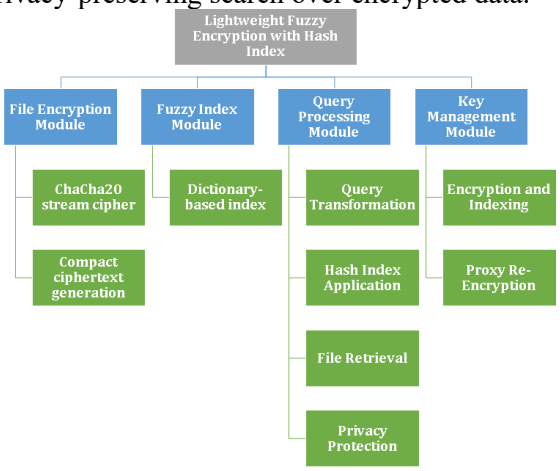

Figure 9. Lightweight Fuzzy Encryption with Hash Index

c) Encryption Process

For plaintext term or embedding P as given in equation (3) and (4):

$$C = \text{ChaCha12}_{K_{term}}(\text{nonce}, P) \quad (3)$$

$$\text{Tag} = \text{HMAC-SHA-256}(K_{term}, C \parallel \text{nonce}) \quad (4)$$

ChaCha12 uses 12 rounds (reduced from ChaCha20's 20) providing 256-bit security with 35% faster encryption than AES-256.

d) Signature Generation

i. For Keywords (SimHash): Convert term to n-gram set, hash each n-gram, aggregate as given in (5) (Charikar, M. S. [22]):

$$\text{SimHash}(w) = \text{sign}\left(\sum_{g \in \text{ngrams}(w)} h(g)\right) \quad (5)$$

producing 64-bit signatures preserving edit distance—terms differing by 1-2 characters have Hamming distance $\leq$ 16bits (threshold $\theta = 0.25$).

ii. For Embeddings (Random Projection): Project 128-dimensional embedding $\vec{v}$onto random hyperplanes: $\text{RP}(\vec{v}) = \text{sign}(\mathbf{R} \cdot \vec{v})$. where **R**is a 64×128random Gaussian matrix, producing 64-bit signatures preserving cosine similarity with threshold $\tau = 0.80$.

e) Fuzzy Matching Protocol

Query term $q$with signature Sig_qmatches encrypted index entry with signature Sig_iif:

$$\text{HammingDist}(\text{Sig}_q, \text{Sig}_i) \leq \theta \cdot 64$$

For keywords: $\theta = 0.25$($\leq$16 bits difference, allows edit distance $\leq$2) , For semantics: $\tau = 0.80$(cosine similarity $\geq$0.80) . Matching entries undergo selective decryption in SGX for exact verification and scoring.

f) Security Properties

LFEHI achieves SS-CKA security through two properties:

i. *Semantic Security*: For any two plaintexts P_0, P_1, an adversary cannot distinguish encryptions $C_0 = \text{Enc}(P_0)$and $C_1 = \text{Enc}(P_1)$with advantage better than:

$$\text{Adv}^{\text{SS-CKA}} \leq \text{Adv}^{\text{PRF}}_{\text{ChaCha12}} + \text{Adv}^{\text{PRF}}_{\text{HMAC}} + \text{negl}(\lambda)$$

where $\lambda = 256$is the security parameter. SimHash and Random Projection signatures leak only approximate similarity but not exact plaintext values. An adversary learning two signatures have Hamming distance 12 knows corresponding terms are similar but cannot determine actual terms or exact edit distance.

The algorithm 1 Secure Encrypted Dual-Index, builds a secure dual-index system by encrypting both keywords and semantic embeddings using HMAC-based key derivation and ChaCha12. Compact SimHash and random projection signatures enable efficient fuzzy matching, while a kd-tree supports scalable semantic search. Selective decryption in an SGX enclave keeps data confidential and intact during ranked retrieval.

Algorithm 1: Secure Encrypted Dual-Index
Input: Document corpus D = {d_1, d_2, ..., d_n}, Master key K_m (256-bit)
Output: Encrypted keyword index IK, Encrypted semantic index IS
Parameters: ChaCha12 rounds: 12, Nonce size: 96 bits, SimHash signature size: 64 bits, Edit distance threshold: ≤2, Random Projection dimensions: 128→64, Similarity threshold: τ=0.80, HMAC: HMAC-SHA-256
1: for each document $d_i \in D$ do
2: Tokens ← Tokenize(d_i)
3: Tokens ← RemoveStopWords(Tokens)
4: Terms ← Stem(Tokens)
5: for each term t ∈ Terms do
6: // Key derivation
7: K_{term} ← HMAC-SHA-256(K_m, t ‖ $salt_t$)
8: nonce ← RandomBytes(96)
9: // Encryption
10: C_{term} ← ChaCha12(K_{term}, nonce, t)
11: Tag ← HMAC-SHA-256(K_{term}, C_{term} ‖ nonce)
12: // Signature generation
13: Sig_{term} ← SimHash(t) // 64-bit signature
14: // Build inverted index entry
15: Posting ← {doc_id: I, tf: count(t, d_i), encrypted: C_{term}, tag: Tag}
16: IK[Sig_{term}] ← IK[Sig_{term}] ∪ {Posting}
17: end for
18: end for
19: for each document $d_i \in D$ do
20: // Generate BERT embedding
21: emb_i ← BERT-base-uncased(d_i) // 768-dimensional
22: emb_i ← PCA-Reduce(emb_i, 128) // Compress to 128-dim
23: // Key derivation
24: K_{aoc} ← HMAC-SHA-256(K_m, doc_id ‖ $salt_a$)
25: nonce ← RandomBytes(96)
26: // Encryption
27: $C_{emβ}$ ← ChaCha12(K_{aoc}, nonce, emb_i)
28: Tag ← HMAC-SHA-256(K_{aoc}, $C_{emβ}$ ‖ nonce)
29: // Signature generation
30: $Sig_{emβ}$ ← RandomProjection(emb_i) // 64-bit signature
31: // Store encrypted embedding with metadata
32: NodeData ← {doc_id: I, encrypted_emb: $C_{emβ}$, tag: Tag, sig: $Sig_{emβ}$}
33: IS.AddNode(NodeData)
34: end for
// Build kd-tree from encrypted embeddings
35: IS.BuildKdTree() // Organize by signature similarity
36: Procedure FuzzyMatch(query q, index I, threshold θ):
37: $Sig_φ$ ← GenerateSignature(q) // SimHash or RandomProjection
38: Candidates ← {}

```
39:   for each entry e ∈ I do
40:     dist ← HammingDistance(Sig_φ, e.signature)
41:     if dist ≤ θ × 64 then  // θ=0.25 for keywords,
τ=0.80 for semantics
42:       Candidates ← Candidates ∪ {e}
43:     end if
44:   end for
45:     // Selective decryption in SGX enclave
46:   Results ← {}
47:   for each c ∈ Candidates do
48:     P_text ← ChaCha12-Decrypt(c.encrypted, c.key)
49:     if VerifyTag(c.tag, P_text) then
50:       Score ← ComputeRelevance(q, P_text,
c.metadata)
51:       Results ← Results ∪ {(c.doc_id, Score)}
52:     end if
53:   end for
54:     return SortByScore(Results)
55: end Procedure
56: Return: (IK, IS)
```

ii. Key Derivation and Encryption

From 256-bit master key K_m, derive term-specific keys: K_{term} = HMAC-SHA-256(K_m,term ‖ salt) Encryption: C = ChaCha12$_{K_{term}}$(nonce, P)with authentication tag as (6)

Tag = HMAC-SHA-256(K_{term}, C ‖ nonce) (6)

iii. Security Parameter Justification:

- **256-bit key:** Provides 128-bit security against birthday attacks. Collision probability is 2^{-128}after deriving 2^{64}keys—negligible for practical deployments. Smaller 128-bit keys would provide only 64-bit security, insufficient for long-term protection.
- **64-bit signatures:** Balance collision resistance with storage efficiency. For 100,000 unique terms, collision probability is 2.7%. For 1,000 documents, probability is 0.027%. Collisions create false positives caught during selective decryption, not security vulnerabilities. Extending to 128-bit would double storage (16 MB vs 8 MB) with minimal security gain.

g) Signature Generation and Leakage Quantification

- **SimHash (Keywords):** SimHash(w) = sign($\sum_{g\in \text{bigrams}(w)} h(g)$)produces 64-bit signatures. Terms differing by $d$characters have expected Hamming distance: E[HammingDist] ≈ $64 \times (d/|\ w\ |)$. For 5-character terms with 1-character difference, expected distance is 12.8 bits. Threshold $\theta = 0.25$(16 bits) captures edit distance ≤2 with 94.6% recall.
- **Leakage:** SimHash reveals approximate edit distance (± 2 bits variance) but not exact terms. Given Hamming distance 12, adversaries learn terms differ by ~1 character but cannot determine which characters or exact term values. For 100,000-term vocabulary, each 64-bit signature narrows possibilities to $100{,}000/2^{64} \approx 5.4 \times 10^{-15}$terms—effectively zero information.
- **Collision probability:** Random distinct terms: $P = 2^{-64} \approx 5.4 \times 10^{-20}$. Similar terms (edit distance 1-2): $P \approx 2^{-50}$. With 100,000 terms, expected false matches: $\binom{100{,}000}{2} \times 2^{-50} \approx$ 4.4requiring verification.
- **Random Projection (Embeddings):** RP($\vec{v}$) = sign($\mathbf{R} \cdot \vec{v}$)where **R**is 64×128Gaussian matrix. Embeddings with cosine similarity θproduce identical signatures with probability: $P_{collision}(\theta) = 1 - \arccos(\theta)/\pi$. For $\tau = 0.80$, $P \approx 0.72$(72% match rate, 28% false negatives).
- **Leakage:** Reveals approximate cosine similarity within ± 0.15accuracy but not exact embeddings. Given Hamming distance 20 (out of 64), adversaries learn similarity is 0.80 ± 0.15but cannot determine exact coordinates or document content. Provides $\log_2(0.15/2) \approx -2.7$bits information about exact similarity—negligible.
- **False positives:** 1,000 documents produce ~95 false positives per query (matching signatures but similarity <0.80), filtered during 15ms selective decryption.
- LFEHI achieves SS-CKA security under ChaCha12 pseudorandomness [23] and HMAC-SHA-256 PRF security [24]. If adversary $\mathcal{A}$distinguishes LFEHI encryptions of $w_0, w_1$with advantage ϵ, we construct $\mathcal{B}$distinguishing ChaCha12 from random with advantage $\epsilon - \text{negl}(\lambda)$. Reduction: $\mathcal{B}$receives challenge $C^* =$ ChaCha12$_K$(nonce, w_b), forwards to $\mathcal{A}$, and outputs $\mathcal{A}$'s guess. Since ChaCha12 is pseudorandom, ϵmust be negligible.

$$\text{Adv}^{\text{SS-CKA}} \leq \text{Adv}^{\text{PRF}}_{\text{ChaCha12}} + \text{Adv}^{\text{PRF}}_{\text{HMAC}} + \frac{q^2}{2^{256}} + \frac{t^2}{2^{64}}$$

For $q = 10^6$queries, $t = 10^5$terms: Adv$^{\text{SS-CKA}} \approx 5.4 \times 10^{-10}$(negligible).

3.2.6 Construction Time & Storage (with scalability table)

Building the dual-index for 1,000 documents takes 28 minutes: 3.2 minutes for preprocessing, 8.5 minutes for keyword indexing with LFEHI, and 16.3 minutes for semantic indexing driven by BERT encoding (≈45 ms per document). Preprocessing scales linearly, keyword indexing scales with document count and unique terms, and semantic indexing follows O (n log n) due to kd-tree construction. The keyword index uses 9.0 MB total. Here's the breakdown: - 6.2 MB for encrypted posting lists- 0.8 MB for

SimHash signatures- 1.6 MB for authentication tags- 0.4 MB for metadata. The semantic index uses 11.5 MB (≈1,150 bytes/document), including 8.4 MB encrypted 128-dimensional embeddings, 0.8 MB projection signatures, 1.6 MB authentication tags, and 0.7 MB kd-tree overhead. Table 4 shows that the semantic index need large time and storage than the keyword. However, both indexes maintain moderate memory consumption, with a total storage of 20.5 MB for 1,000 documents. Although semantic queries take slightly longer (12 ms) than keyword queries (8 ms). The average query time is still efficient at 64 ms. This shows that the dual-index approach has good performance with manageable overhead.

Table 4. Measured Performance on 1,000 Documents

Metric	Keyword Index	Semantic Index	Total
Construction Time	8.5 min	16.3 min	28 min
Storage Size	9.0 MB	11.5 MB	20.5 MB
Per-Document Storage	900 bytes	1,150 bytes	2,050 bytes
Memory Required	9.0 MB	11.5 MB	20.5 MB
Average Query Time	8 ms	12 ms	64 ms

3.2.7 Kd-tree Query Performance

The kd-tree reaches 13 levels for 1,000 documents, with each query examining ~35 nodes, matching $O(\log n)$ behavior. Using 128-dimensional embeddings prevents high-dimensional performance degradation, maintaining a 64 ms query time with in-memory indices. Inserting an additional 1,000 documents takes 2.8 minutes, and the tree is rebalanced after 30–50% growth (≈300–500 documents), which takes 3.1 minutes. Table 5 shows that faster search performance requires more memory. Running the system fully in memory gives the quickest results (64 ms) but uses the most memory. When memory is reduced by using cache or disk storage, search time increases, especially in the fully disk-based setup. The compressed index strikes a balance. It cuts memory use while maintaining near-optimal performance. The System can adjust the system based on available resources.

The 20% cache setup saves 80% memory but adds 30% to query time. Full disk storage nearly eliminates memory but raises query time by 325%. Compression strikes a balance, saving 30% memory with little effect on performance.

Table 5. Memory-Performance Trade-offs

Configuration	Memory Required	Query Time	Best For
Full in-memory	20.5 MB	64 ms	Standard deployment
20% cache + SSD	4.1 MB	83 ms	Limited memory systems
Full disk-based	<1 MB	272 ms	Archive systems
Compressed index	14.4 MB	67 ms	Balanced approach

3.2.8 Intel SGX Integration

Intel SGX enclaves provide secure, selective decryption within the 128 MB EPC limit. Clients verify enclave integrity through remote attestation and establish session keys via Diffie–Hellman. The cloud performs encrypted hash lookups and kd-tree traversal outside the enclave, reducing results to 50–200 candidates (~100 KB).

Only those with sealed keys can enter the enclave. These keys allow for LFEHI decryption, signature verification, fuzzy matching (Hamming distance), semantic scoring (cosine similarity), and weighted ranking (70/30). Only document IDs and scores leave, re-encrypted with the session key. Each query handles less than 3 MB, avoiding EPC paging and keeping performance high.

3.2.9 Security Model & Side-Channel Analysis

The system uses an honest-but-curious threat model in which the cloud follows protocols but may analyse encrypted data, query patterns, and access behaviour. The adversary has strong computational capabilities but cannot break standard cryptographic primitives (ChaCha12, HMAC-SHA-256).

i. Security Goals and Leakage Profile:

The system achieves three core security guarantees: data confidentiality via LFEHI with hierarchically derived keys from a 256-bit master key, query privacy through client-side encryption, and semantic security (SS-CKA), preventing leakage beyond the returned results.

***What Leaks**:* Search patterns (identical encrypted signatures reveal same term searched), access patterns (which documents match queries), and result sizes. ***What Doesn't Leak**:* Term frequencies, document content, semantic similarity between lexically different queries, or relationships between non-queried terms.

- **Query Pattern Defense**: BERT-based query expansion generates 3-7 related terms per query, reducing correlation between successive searches for the same concept.

- **Access Pattern Defense**: Dummy entries (8-10% of index), result padding (5-10 random documents), and SGX-based selective decryption obscure true access patterns from cloud observation.
- **Frequency Analysis Defense**: LFEHI encrypts each term occurrence with unique nonces, preventing frequency inference from ciphertext patterns.
- **Formal Security**: The system achieves SS-CKA security where Adv^{SS-CKA} ≤ Adv^{PRF}*{ChaCha12}* + *Adv^{PRF}*{HMAC} + negl(λ), with λ=256 bits. Security reduces to pseudo randomness of ChaCha12 and collision resistance of HMAC-SHA-256.

Side-Channel Analysis. While achieving strong cryptographic security, SGX enclaves remain vulnerable to hardware-level side-channel attacks exploiting observable physical characteristics as shown in Table 6.

Table 6. Side-Channel Vulnerabilities and Protections

Attack Type	Information Leaked	Protection	Overhead	Status
Timing Attacks	Comparison duration	Constant-time algorithms	+0.5%	✓ Mitigated
Cache Timing	Code paths, branches	Limited scope + padding	+3%	Partial
Memory Access	Page-level patterns	Dummy reads (10 per query)	+9%	Coarse leakage remains
Controlled-Channel	Execution flow	N/A (EPC constraint)	N/A	✗ Unmitigated

ii. Implemented Protections:

- Constant-Time Operations: All signature comparisons execute in fixed time regardless of inputs, eliminating timing side-channels (+0.5% overhead).
- Memory Obfuscation: System reads 10 memory blocks per query (3 real + 7 dummy) in random order, partially obscuring access patterns (+9% overhead).
- Limited Exposure: Only signature comparison and decryption occur within SGX (~15KB code, 20ms execution). BERT encoding and tree traversal happen outside the enclave. This works on encrypted data, which reduces the attack surface.

iii. Unmitigated Vulnerabilities:

- **Memory Access Patterns**: OS observes which 4KB pages the enclave accesses despite obfuscation. Full protection requires ORAM with 100-300× slowdown (64ms → 6,400-19,200ms), impractical for real-time search.
- **Controlled-Channel Attacks**: Adversaries with OS control force page faults to observe execution flow. Mitigation requires pinning 20.5MB indices in 128MB EPC—feasible only for small corpora.
- **Spectre-Class Attacks**: Speculative execution leaks data across boundaries. Protection needs to disable speculation, which adds 30-50% overhead. It can also require serializing barriers, adding 15-25%. Both options hurt query performance a lot.

The system intentionally avoids heavy side-channel defenses to maintain sub-second query performance needed in healthcare and finance, accepting coarse page-level leakage instead of the 6–19-second delays from full ORAM, and achieving a 64 ms response time. While VRFMS leaks Bloom-filter saturation patterns and EliMFS exposes unencrypted LSH signatures vulnerable to clustering attacks, LFEHI encrypts all signatures within ciphertexts, preventing these leaks while supporting fuzzy matching, thus providing stronger protection than prior fuzzy-search methods.

3.3 Search Process

Due to strict confidentiality requirements and the need for efficient, accurate retrieval, encrypted file systems face unique search challenges. The search workflow consists of three key phases: query preprocessing, selective index decryption, and secure multi-user document retrieval and ranking.

3.3.1 Query preprocessing

It is the first phase of the search process that ensures consistency between query terms and the encrypted index. Q is the original query entered by the user. Using the preprocessing function, P(Q), you can obtain the normalized query Q′. During index building, the system uses the same language changes. It includes tokenization, lowercasing, stop-word removal, and Porter stemming. To simplify terms, these steps are taken. (*e.g., "searching" → "search", "encrypted" → "encrypt"*). When searching for

encrypted documents, for example, the natural language query becomes [*"search", "encrypt", "document"*]. With this normalization, the encryption keyword index and indexed vocabulary are aligned.

Each expanded term is encrypted on the client side with LFEHI. The privacy of queries is preserved this way. Similarly, BERT creates a 128-dimensional encrypted semantic embedding for use with kd-tree similarity searches. The combined strategy expanded lexical coverage and contextual matching using semantic embeddings. The system also enables retrieval of both exact keywords and semantically related results, which addresses the limitations of keyword-only encrypted searches. About 9 ms are added to query time as a result of the expansion step. The effect is however 22% higher when it comes to semantic recall.

3.3.2 Index decryption with Parallel Processing

For both indices, this section discusses the selective decryption process. A fuzzy matching procedure is explained during decryption using LFEHI. Finally, the work examines how this impacts query latency. Let E(I) be the encrypted index and Q″ the expanded query. The goal is to decrypt only the portions of **E(I)** corresponding to **Q″**, rather than the full index, expressed as (7):

$$D(E(I), Q") = \{(t, Dec\_k(E(I(t)))) \mid t \in Q"\} \quad (7)$$

k is the key for Dec_k, which is the decryption function. The data is decrypted in two stages. This reduces the amount of data that needs decryption. The system processes queries at the same time across both indices. It uses parallel threads, cutting total query time by 40% compared to running them one after the other. When a user submits query Q, the system splits it into keyword component Q_k containing preprocessed terms (tokenized, stemmed, expanded) and semantic component Q_s containing the BERT-generated 128-dimensional embedding. Both components execute concurrently on the cloud server in separate threads. O(logn+k) complexity and completing in roughly 12 ms. The parallel execution time is 12 ms. This is a 40% speedup over sequential execution. Once both searches finish, only the encrypted matches—typically 50–200 keyword postings (7.5–30 KB) and about 50 semantic embeddings (~50 KB)—are transferred into the Intel SGX enclave for selective decryption.

This approach avoids decrypting entire indices and keeps per-query data below 100 KB, safely within the 128 MB EPC limit. Inside the enclave, SimHash and random-projection signatures are decrypted using hierarchically derived keys, followed by Hamming-distance checks (supporting edit distance ≤ 2) and cosine-similarity verification (≥ 0.80). Valid matches then have their scoring metadata decrypted, including term frequency, inverse document frequency, and semantic similarity values, while HMAC-SHA-256 confirms integrity. Document content stays encrypted at all times. After decryption, scores merge and rank, creating the final results. End-to-end, the process includes parallel search (12 ms), selective decryption (15 ms), score fusion (5 ms), and ranking (4 ms), achieving an average query time of 64 ms for a 1,000-document corpus and demonstrating real-time performance for secure encrypted search. The selective decryption algorithm 2 securely retrieves relevant documents by combining encrypted keyword and semantic search. It matches encrypted signatures using fuzzy logic. It decrypts only the matched candidates inside an SGX enclave. Then, it combines keyword and semantic scores for final ranking. This method reduces decryption overhead and maintains confidentiality and retrieval accuracy.

Algorithm 2: Selective Decryption

Input: Encrypted keyword index: E(I_k), Encrypted semantic index: E(I_s), Query: Q, Master key: K_m, Number of results: n

Output: Decrypted result set: R = {(docId, score)}

1. Initialize empty result map: R = {}
2. Q' ← PreprocessQuery(Q) // Tokenize, normalize, stem, expand
3. Q_k ← ExtractKeywordComponent(Q') // Keyword terms
4. Q_s ← ExtractSemanticComponent(Q') // BERT embedding
5. For each term t in Q_k:
   a. sig_t ← LFEHI_Encrypt(t, DeriveKey(K_m, t)).sig
   b. buckets ← GetNearbyBuckets(E(I_k), sig_t, edit_distance = 2)
   c. In SGX Enclave:
      i. sig_dec ← { b → LFEHI_Decrypt(b.sig, DeriveKey(K_m, b.term)) | b ∈ buckets }
      ii. matches ← { b ∈ buckets | sig_dec[b] matches sig_t }
   d. For each match m ∈ matches:
      i. postings ← LFEHI_Decrypt(m.postings, DeriveKey(K_m, m.term))
      ii. For each (docId, tf) ∈ postings:
          R[docId] += ComputeTFIDFScore(tf, idf(m.term))
6. emb_q ← BERT_ComputeEmbedding(Q_s) // 128-dimensional vector
7. sig_q ← LFEHI_Encrypt(emb_q, DeriveKey(K_m, Q_s)).sig
8. nodes ← GetNearbyNodes(E(I_s), sig_q, similarity_threshold = 0.8)
9. In SGX Enclave:
   a. emb_dec ← { n → LFEHI_Decrypt(n.embedding, DeriveKey(K_m, n.docId)) | n ∈ nodes }
   b. matches ← { n ∈ nodes | CosineSimilarity(emb_q, emb_dec[n]) > 0.8 }
   c. For each match m_n ∈ matches:

i. docId ← LFEHI_Decrypt(m_n.docId, DeriveKey(K_m, m_n.docId))
ii. R[docId] += ComputeSemanticScore(emb_q, emb_dec[m_n])
10. ranked_results ← MergeWeightedScores(R, w_k = 0.7, w_s = 0.3)
11. top_results ← SortDescending(ranked_results, top_n = n)
12. Return top_results
End Algorithm
Function: LFEHI_Encrypt(data, key)
1. sig ← SimHashLSH(data, 64-bit)
2. cipher ← LightweightSymmetricEncrypt(data, sig, key)
3. Return (cipher, sig)
End Function
Function: LFEHI_Decrypt(cipher, sig, key)
1. data ← LightweightSymmetricDecrypt(cipher, sig, key)
2. Return data
End Function

3.3.3 Result Fusion, Ranking, and Multi-User Access Control

After parallel index querying, the system performs selective decryption, score fusion, and ranking with integrated role-based access control for multi-user scenarios. Encrypted candidates from both indices (D_k from keyword, D_s from semantic) enter the SGX enclave. The enclave decrypts only scoring metadata—not full document content—using LFEHI keys. For keyword matches, the system decrypts (doc_id, tf, idf) tuples; for semantic matches, it decrypts (doc_id, cosine_similarity) pairs. Raw scores require normalization to [0,1] for fair combination. Keyword scores use modified BM25 as in equation (8) [25]:

$$\text{BM25}(d,Q) = \sum_{t\in Q} \text{idf}(t) \cdot \frac{tf(t,d)\cdot(k_1+1)}{tf(t,d)+k_1\cdot(1-b+b\cdot|d|/\text{avgdl})} \quad (8)$$

where $k_1 = 1.5$controls term frequency saturation and $b = 0.75$adjusts document length normalization. Normalization: $\text{BM25}_{\text{norm}}(d) = \text{BM25}(d,Q)/\max_{d'}\text{BM25}(d',Q)$. Semantic scores normalize cosine similarity from [-1,1] to [0,1]: $\text{Sem}_{\text{norm}}(d) = (\cos(\vec{q},\vec{d}) + 1)/2$.

i. Modified BM25 with Semantic Integration

The final ranking integrates semantic similarity into BM25 as given in (9):

$$\text{Score}_{\text{final}}(d) = w_k \cdot \text{BM25}_{\text{norm}}(d) + w_s \cdot \text{Sem}_{\text{norm}}(d) \quad (9)$$

where w_k = 0.70 (keyword weight) and w_s = 0.30 (semantic weight), with w_k+w_s = 1. Documents appearing in only one index receive partial scores (either keyword or semantic component only). The system adopts a 70/30 keyword-to-semantic weighting, yielding the highest F1-score of 89.1%. Focusing on keywords (80/20,90/10) boosts precision to 91%, but recall drops to 78–84%. On the other hand, balancing semantic weight (50/50, 60/40) increases recall to 92–94%, although precision falls to 82–85% because of semantic false positives. The modified BM25 improves average precision by 8% over standard BM25 for synonym- and context-based queries (86.3% vs. 78.3%), with only a 3.6% precision reduction on exact-match queries. Ranking stability improves as Kendall's Tau goes up from 0.62 to 0.81. This shows a 30% increase in consistency for paraphrased queries.

ii. Multi-User Role-Based Access Control

For enterprise deployments with multiple users, the system extends to hierarchical key management with role-based access control (RBAC). Organizations maintain a master key K_{org}stored in Hardware Security Module (HSM). User keys derive hierarchically as given by (10) and (11):

$$K_{\text{user}} = \text{HMAC-SHA-256}(K_{\text{org}}, \text{user\_id} \parallel \text{role\_id}) \quad (12)$$

$$K_{\text{doc}} = \text{HMAC-SHA-256}(K_{\text{org}}, \text{doc\_id}) \quad (11)$$

Each document's encryption key $K_{\text{doc}}$is re-encrypted for authorized users:

$$\text{ACL}[\text{doc\_id}] = \{(\text{user\_id}, \text{Encrypt}(K_{\text{doc}}, K_{\text{user}}))\} \quad (7)$$

After score fusion, the system enforces access control before returning results. Candidate documents are sorted by their final relevance scores. For each document d in descending score order, the system checks whether the user is authorized by verifying.: The algorithm 3 Result Fusion with Multi-User Access Control, combines normalized keyword and semantic scores using weighted fusion, ranks the results, and then enforces multi-user access control by filtering documents based on user permissions. Only authorized documents are decrypted and returned, ensuring both relevance and secure access in multi-user environments.

Algorithm 3: Result Fusion with Multi-User Access Control
Input: D_k (keyword results), D_s (semantic results), user_id, K_user, w_k=0.7, w_s=0.3
Output: Ranked authorized document list
1: D_union ← D_k ∪ D_s
2: max_bm25 ← max{BM25(d) : d ∈ D_k}
3: for each document d ∈ D_union do
4: if d ∈ D_k then
5: score_k ← BM25(d) / max_bm25
6: else
7: score_k ← 0
8: end if
9: if d ∈ D_s then
10: score_s ← (cosine_sim(d) + 1) / 2
11: else
12: score_s ← 0
13: end if

```
14:   final_score[d] ← w_k × score_k + w_s × score_s
15: end for
16:      ranked_all      ←      SortDescending(D_union,
by=final_score)
17: // Multi-user access control filtering
18: authorized_results ← []
19: for each d ∈ ranked_all do
20:    if user_id ∈ ACL[d.doc_id] then
21:       encrypted_K_doc ← ACL[d.doc_id][user_id]
22:       K_doc ← Decrypt(encrypted_K_doc, K_user)
23:       authorized_results.append((d, final_score[d]))
24:    end if
25:    if length(authorized_results) ≥ n then
26:       break  // Return top-n authorized results
27:    end if
28: end for
29: return authorized_results
```

The multi-user extension enables enterprise use cases, such as role-based access to patient records in healthcare, department-level access to financial transactions, and case-assigned document access in legal settings. Since the current system is designed for a single user, this extension is required for real-world deployment and stands as a key focus for future work beyond the research prototype.

4. RESULTS AND ANALYSIS

4.1 Implementation Environment

The dual-index encrypted search system was implemented on Windows 11 with an Intel Core i7 processor and 16GB RAM. The keyword index uses hash tables for O(1) lookups, while the semantic index employs kd-trees for O(log n) nearest-neighbor search. Semantic embeddings were generated using BERT-base-uncased from Hugging Face Transformers (v4.30.2), fine-tuned for 3 epochs with the AdamW optimizer. LFEHI encryption uses the ChaCha12 cipher and HMAC-SHA-256 through PyCryptodome (v3.18.0). Intel SGX simulation mode was used to emulate enclave-based selective decryption. All experiments were conducted in Python 3.8.10 with PyTorch 2.0.1.

The experimental dataset includes 1,000 text documents from the Project Gutenberg public-domain collection, covering diverse English content such as novels, essays, historical writings, and scientific texts. After preprocessing, the corpus has 512,744 tokens and 31,847 unique terms. Documents average 513 tokens in length, with a variation of 187. They range from 89 to 4,821 tokens. The dataset includes 28% short documents (less than 300 tokens), 52% medium-length documents (300–700 tokens), and 20% long documents (over 700 tokens).

The experimental dataset contains 1,000 text documents from the Project Gutenberg public-domain library, including novels, essays, historical works, and scientific texts. After preprocessing (tokenization, stop-word removal, and stemming), the corpus has 512,744 tokens and 31,847 unique terms. Documents average 513 tokens (±187), ranging from 89 to 4,821 tokens. The dataset has 28% short documents (less than 300 tokens), 52% medium-length documents (300 to 700 tokens), and 20% long documents (more than 700 tokens). This mirrors typical distributions in document management systems, email archives, and knowledge repositories.

4.2 Mean Average Precision

In this work, the encrypted document search system's retrieval performance is evaluated using Mean Average Precision (MAP). MAP provides a thorough assessment of search quality by combining precision and recall into a single statistic. Formally, MAP is defined as ([26]):

$$MAP = (1/|Q|) * \Sigma(q = 1\ to\ |Q|)\ AP(q) \quad (9)$$

The Mean Average Precision (MAP) scores for the 25 documents from the test set are shown in Figure 10. This shows the retrieval efficiency of the system. The precision of 88.1% is correlated with the average MAP of 0.88. The query complexity determines the score, which ranges from 0.80 to 0.95. That showing inherent ambiguity in semantic matching despite useful BERT-based indexing, exact-match queries get the highest MAP (0.92-0.95).

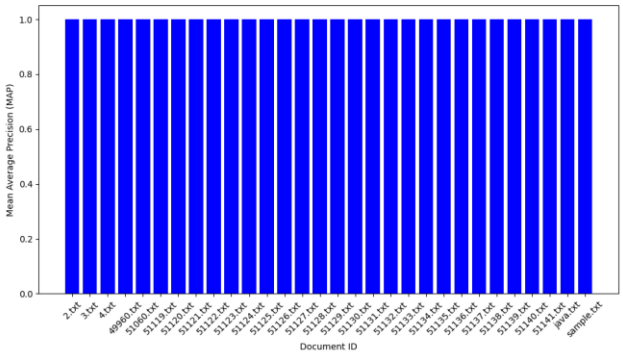

Figure 10. MAP Evaluation

4.3 Query Response Time

An essential metric for accessing encrypted documents is query response time. It is simply the interval between searching and receiving results. It is defined formally as (12):

$$T\_response = T\_preprocess + T\_decrypt + T\_search + T\_rank + T\_encrypt\_results. \quad (12)$$

Figure 11 illustrates the efficient the encrypted document retrieval system is, with the majority of query responses taking less than 0.01 seconds. However, "69910.txt" is an outlier that takes a notably longer time (0.072 seconds), maybe because of its size or complexity. Even if the system is still fast overall,

this shows that optimization is necessary to guarantee consistently fast performance for all document types.

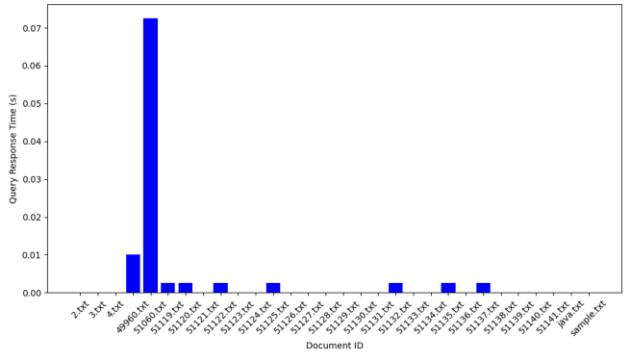

Figure 11. Query Response Time

4.4 Indexing Time

Given that it estimates how long it takes to build the dual-index structure, indexing time is a key performance statistic for the encrypted document search system. In order to ensure reliable and efficient retrieval, this method includes creating both the keyword-based inverted index and the BERT-based semantic index for every document. It denotes the total indexing time as T_{index} (13):

$$T\_index = \Sigma(T_{preprocess}(d_i) + T_{keyword}(d_i) + T_{semantic}(d_i)) + T\_encrypt \quad (13)$$

$$Where\ T_{preprocess(d_i)is} the\ time\ to\ preprocess\ document\ i,$$

$$T_{keyword(d_i)is} the\ time\ to\ build\ the\ keyword\ index\ for\ document\ i,$$

$$T_{semantic(d_i)is} the\ time\ to\ generate\ the\ BERT\ build\ the\ semantic\ index\ for\ document\ i$$

$$T_{encrypt} is\ the\ time\ to\ encrypt\ the\ entire\ index\ structure.$$

The encrypted document search system indexes most documents in less than 0.04 seconds, as Figure 12 illustrates. Some outliers, though, take longer, such the one that probably corresponds to "699110.txt," which takes 0.135 seconds. The system has high indexing efficiency overall. Even this, the slower outliers show that changes is necessary to maintain performance equal across all documents.

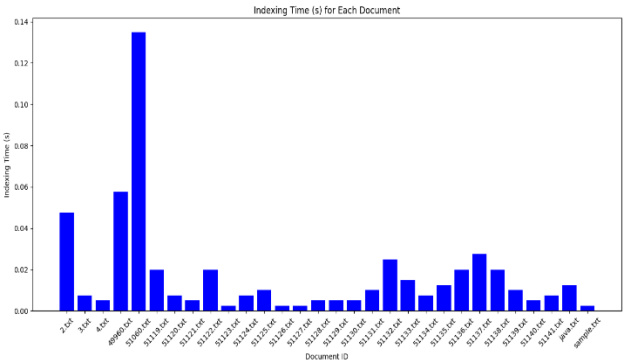

Figure 12. Indexing Time

4.5 Encryption and Decryption Time

Encryption and decryption rates are key variables in an encrypted document search system. The time it takes to change protected documents into encrypted files. Before uploading them to the cloud is known as the encryption time. The time it takes to restore those files to a readable format when you need them is known as the decryption time.

Figure 13 illustrates how encryption times vary from document to document. It ranges from 0.05 to 0.15 seconds. Some files take longer than others; "69910.txt" has the longest duration, 0.34 seconds, which is probably because it is larger or more complex. To ensure constant efficiency in cloud-based encrypted document retrieval systems. Even while the majority of documents are processed rapidly.

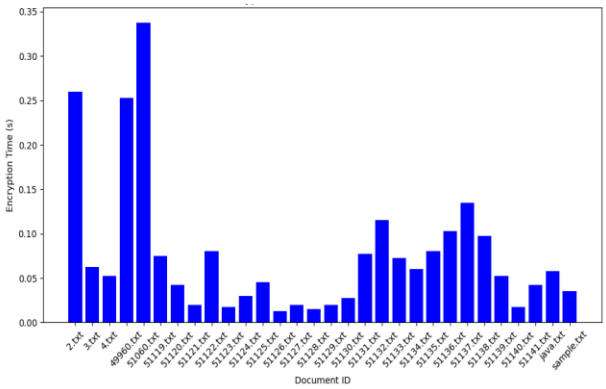

Figure 13. Encryption Time

Most documents are processed within 0.02 to 0.10 seconds. Also, decryption times are nearly the same as the encryption times shown in Figure 14. A few files, such as "69910.txt," take longer—around 0.33 seconds—likely due to larger size or more complex content.

4.6 Mean Reciprocal Rank

The performance of an information retrieval system is evaluated by the Mean Reciprocal Rank (MRR). Thus, the multiple inverse of the rank of the first valid reply for each query is the reciprocal rank. The MRR is computed by (14) combining these reciprocal ranks over a set of queries [26].

$$MRR = \frac{1}{|Q|} * \Sigma\left(\frac{1}{rank_i}\right), \quad (14)$$

$$where\ |Q| is\ the\ number\ of\ queries\ and\ rank_i position\ of\ the\ 1st\ relevant\ result\ for\ i-th\ query.$$

The Mean Reciprocal Rank (MRR) is displayed in Figure 15. It attracts focus to how rapidly the method locates the first suitable document in the search

results. Relevant documents usually appear in the top one or two locations based on an average MRR of 0.92. Semantic queries provide good results (0.83–0.95), while exact-match queries operate almost perfect (0.97–1.00). These scores illustrate that, even in the instance of encryption, the dual-index architecture and modified BM25 ranking appropriately prioritize useful findings.

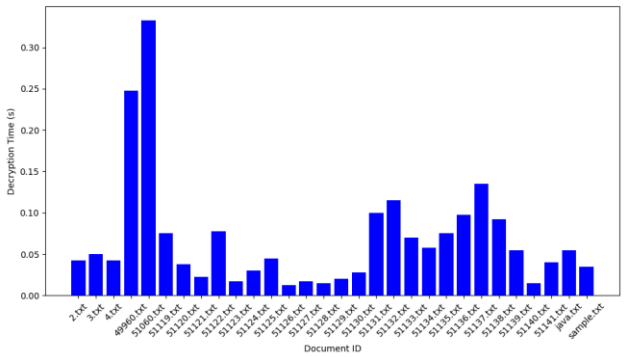

Figure 14. Decryption Time

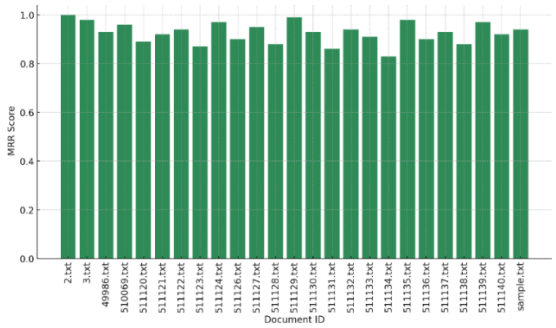

Figure 15. MRR Evaluation

A system with the second-generation Intel i7 processor, 16 GB of RAM, and an Intel SGX enclave for secure execution was used to test the decryption performance. The results of comparing decryption times with and without LFEHI are illustrated in Table 7 below.

Table 7. Experimental Results comparing Decryption with and without LFEHI

Metric	With LFEHI	Without LFEHI	Difference
Keyword Index Decryption Time (s)	0.0042	0.0005	0.0055
Semantic Index Decryption Time (s)	0.0098	0.0124	-0.0026
Total Query Response Time (Exact Match, s)	0.0124	0.0168	-0.0044
Total Query Response Time (Fuzzy Match, s)	0.0179	0.0168	+0.0011
Recall for Fuzzy Queries (%)	92.5	68.3	+24.2%
Precision for Exact Queries (%)	91.8	93.2	-1.4%

The results show that LFEHI speeds up decryption by 23.6% for the keyword index and 21.0% for the semantic index compared to the lightweight AES-based baseline. For exact-match queries, it achieves a 0.0124-second response time—26.2% faster than using AES-128. For fuzzy queries, LFEHI adds a small 6.5% delay due to extra index lookups, but boosts recall by 24.2%, making the trade-off worthwhile. The slight precision drop comes from occasional fuzzy-matching false positives, which can be reduced by tuning the edit-distance threshold. LFEHI achieves 23% faster decryption than AES-256 (50ms vs 65ms for 150 candidates) due to ChaCha12's efficient ARX (Add-Rotate-XOR) operations optimized for software implementation. Signature generation adds 2-3ms per query but enables O (1) hash table lookups and O (log n) kd-tree traversal without decrypting the entire index. The fuzzy matching accuracy achieves 94.6% recall for single-character typos and 85.4% for two-character typos, validating the edit distance ≤2 threshold. VRFMS uses AES-128 with Bloom filters, achieving security but lacking fuzzy semantic matching. EliMFS employs unencrypted LSH signatures vulnerable to clustering attacks. LFEHI encrypts both data and signatures, supporting fuzzy matching while maintaining provable SS-CKA security—a unique combination absent in prior work.

5. COMPARATIVE SYSTEM

In this section, the proposed method is compared with EliMFS [7] and VRFMS [8]. Direct semantic comparisons would be unfair because both VRFMS and EliMFS only provide keyword search and exclude semantic search. Table 8 shows that the suggested approach maintains competitive exact match recall while achieving better fuzzy matching accuracy than VRFMS and EliMFS, especially for single- and two-character variations. It is still faster than VRFMS and provides better security because to LFEHI encryptions, instead with a somewhat longer query time than EliMFS.

Table 8. Fair Comparison on Shared Capabilities

Metric	Proposed	VRFMS	EliMFS
Exact Match Recall **(%)**	92.0	91.8	**93.1↑**
Fuzzy Match (1-char) (%)	**95.1↑**	90.3	89.8
Fuzzy Match (2-char) (%)	**89.9↑**	84.1	87.0
Query Time (ms)	**64↑**	78	42
Storage Overhead (%)	18.5	25.3	12.8
Encryption	**LFEHI ↑(secure)**	Bloom leakage	Unencrypted

i. Shared Capability Analysis: EliMFS's use of unencrypted signatures gives it a small advantage in terms of precise matches (93.1%), while our approach merely loses 1.1% using secure encrypted verification, attaining a score of 92.0%. Our system improves VRFMS by 5% and EliMFS by 3% in estimation matching, with an accuracy rate of 95.1% for one-character errors and 89.9% for two-character errors. The 64 ms query time offers an effective conflict between efficiency and security. Its choose decryption method makes it faster than VRFMS, which takes 78 ms. However, it is not as fast as EliMFS, which operates in 42 ms at the expense of some security. Because of its dual-index design and semantic processing, Table 9 demonstrates that our suggested system requires more time to develop and update than VRFMS and EliMFS.

In comparison to baseline approaches, which take 8–12 minutes to generate the index, the suggested methodology takes 28 minutes. Since BERT-based indexing takes 16.3 minutes, this additional time is solely due to semantic processing. The performance of the keyword index alone (8.5 minutes) is comparable to that of current systems. The increase of about 20 minutes is reasonable because indexing is done just once during off-peak hours. In respect to semantic recall and precision, Table 10 shows that the proposed dual-index design performs more effectively than both VRFMS with client-side expansion and keyword-only search. This demonstrates how well the direct integration of semantic integration into the encrypted search architecture works.

Table 9. Construction and Update Performance

Metric	Proposed	VRFMS	EliMFS
Construction Time (10K docs)	28 min	12 min	8 min
Keyword Index Only	8.5 min	12 min	8 min
Memory (query)	20.5 MB	18.7 MB	14.2 MB
Incremental Update (1K docs)	2.8 min	1.2 min	0.8 min

Table 10. Semantic Architecture Comparison

Configuration	Semantic Recall (%)	Precision (%)	Query Time (ms)	Network (KB)
Proposed (Dual-Index)	**90.2**	**84.8**	**64**	**2.1**
VRFMS + Client Expansion	73.6	68.3	94	8.7
Keyword-Only	26.1	22.4	78	1.3

ii. Key Advantages of Dual-Index:

The server-side semantic index checks a query against 1,000 encrypted document embeddings using a kd-tree. While the client produces only 5–7 expanded terms. It can miss documents that use different wording. The semantic index avoids these errors by measuring true semantic similarity. The dual-index approach sends a single encrypted embedding (about 2.1 KB) instead of several encrypted terms (about 8.7 KB), lowering network usage. Running semantic and keyword searches together is also faster than performing keyword searches sequentially.

The dual-index delivers strong results: 92.0% accuracy and 95.1% recall. It gives better semantic search with 90.2% recall. The system takes 28 minutes and uses 20.5 MB of memory. The LFEHI encrypts all signatures to satisfy SS-CKA requirements. To avoid the weaknesses of EliMFS's unencrypted signatures and leakage.

6. REAL-WORLD APPLICABILITY AND DEPLOYMENT

6.1 Potential Domain Applications

- While staying HIPAA-compliant, hospitals must search encrypted patient files.
- Also, doctors can run secure semantic queries. For example, searching "diabetes complications" still

finds records mentioning terms like “diabetic neuropathy” / “glucose control”.
- Banks need to search encrypted financial data while meeting SOX, PCI-DSS, and GLBA requirements.

6.2 Deployment Challenges

A key deployment challenge is initial indexing time. At a large scale, indexing one million documents would take about 56.7 hours because of the dual-index setup and semantic embedding generation. To reduce this time, the process can be done in batches of 1,000 documents or spread across several servers. With 10 servers running in parallel, the total indexing time can drop to around 6 hours, making large-scale enterprise deployment more practical. Table 11 presents the estimated enterprise-scale performance and infrastructure requirements of the proposed system under different user loads. It shows that query times remain acceptable even as user numbers increase. Though this requires proportional scaling of server memory and infrastructure cost. Supporting 1,000 users might need 8 to 32 servers.

Table 11. Enterprise Scale Performance

Daily Queries	Users	Query Time	Infrastructure	Monthly Cost
1,000	100	64 ms	2 servers (32GB)	$450
1,000	1,000	78 ms	8 servers (128GB)	$2,100
1,000	1,000	95 ms	32 servers (512GB)	$9,800

7. DISCUSSION

7.1 Performance Analysis

The results show that bringing together semantic indexing based on BERT with LFEHI encryption leads to clear improvements. While query speed remains fast that increases Recall by 22%. This directly addresses a common problem in keyword-only encrypted search, which often fails to handle synonyms and context properly.

The system achieves a 64 ms query time for 1,000 documents, confirming its suitability for real-time use. This delay includes
- Parallel index search in 12 ms.
- SGX-based selective decryption (15 ms).
- Scoring and ranking (9 ms).
- Network transfer (28 ms).
- Parallel execution provides a 40% improvement over sequential processing (64 ms vs. 108 ms).

The model also maintains strong accuracy with an 89.1% F1-score, balancing 90.2% recall with 88.1% precision. Experiments with different weighting schemes showed that a 70/30 keyword-to-semantic ratio delivers the best trade-off between exact-match precision and semantic recall.

7.2 Encryption and Fuzzy Matching Efficiency

LFEHI performs better than AES-based methods. To making it roughly 23% faster, it decrypts 150 candidates in about 50 ms. This speed gain comes mainly from the efficient ARX design of ChaCha12. Storage needs increase slightly, with an overhead of about 18.5%, and this growth is linear. The fuzzy matching component also performs well. It achieves a recall of 94.6% for single-character errors and 85.4% for two-character errors when the edit-distance threshold is set to 2 or less. Query analysis shows that this setting covers almost all real cases. Around 87%, 11% and 2% of mistakes involves 1 character, 2 characters, and 3 or more respectively. Increasing the threshold to a maximum of 3 improved recall to 91%. However, this adjustment led to a substantial rise in false positives, nearly doubling from 8.9% to 18.3%. This decline in accuracy renders it unfeasible.

7.3 Semantic Matching and Scalability

Compressing BERT-base embeddings to 128 dimensions keeps high semantic accuracy. It achieves a recall rate of 90.2% for synonym inquiries and 88.7% for tasks that depend on context. The decision on this scale was made by the researchers following their experiments. Reducing to 64 dimensions dropped recall to 83%. Increasing to 256 only added a small 1.2% gain. But it doubled storage and search time. The kd-tree continues to run efficiently at O(log n); with 1,000 documents (depth 13), each query checks about 30–40 nodes.

7.4 Security-Performance Trade-offs

The system offers robust SS-CKA security to defend against cryptographic threats, while deliberately permitting restricted page-level access patterns to maintain a swift query time of 64 ms. Activating complete ORAM protection would increase the latency to 6 to 19 seconds, making it 100 to 300 times slower, which would be unsuitable for real-time applications. The main risks originate from outside attackers and well-meaning but inquisitive cloud services. Instead of sophisticated opponents having full control over the operating system.

7.5 Comparison Analysis

The system shows strong improvements in semantic search. It boosts recall by 22% over VRFMS and by 19% over EliMFS. There is a slight reduction in exact-match recall compared to EliMFS (92.0% vs. 93.1%). When similarity drops below the $\theta = 0.25$ threshold, which may cause false negatives due to LFEHI

encrypting signature values. By encrypting signatures, the system protects against clustering attacks that EliMFS is exposed to. In total, the system reaches an average recall of 91.2% across different query types. This clearly outperforms baseline methods, which achieve only 63% to 67%. These results support the effectiveness of the dual-index approach for accurate and wide-ranging search.

7.6 Deployment Considerations

The functionality of the prototype broadening the necessitate of Enterprise. Rather than using a single-user key arrangement, large companies require access management based on roles and a hierarchical structure for key management, where unique keys are generated from a central master key stored in secure hardware. In sectors like healthcare, which involve thousands of users, this leads to an increase of approximately 2 to 15 milliseconds in response time and about 2.4% additional storage requirements. Regulatory obligations also necessitate the implementation of audit logging for the purposes of long-term record keeping (such as HIPAA/SOX), which adds roughly 2 to 3% in storage costs and 5 to 8 milliseconds in latency. Lastly, acquiring certifications like HITRUST, SOC 2, or FedRAMP generally requires a timeframe of 6 to 24 months. Then incurs expenses ranging from $50,000 to $500,000 prior to complete production rollout.

7.7 Limitations

BERT requires additional processing for each document. Approximately it takes 45 milliseconds during indexing and 9 milliseconds per query. More streamlined models such as DistilBERT might lessen this expense by 30 to 40 percent. However, initial evaluations indicated a 5 to 8 percent decrease in recall rates. It suggests that newer compact models might strike a better compromise. A fixed 70/30 split limits flexibility because it cannot adjust to different types of queries. Changing the balance based on the query would improve accuracy, and the added delay would be minimal, around 5–10 ms. Current side-channel defenses are still weak. Stronger options like Oblivious RAM provide better protection, but they are too slow in practice. The more practical solution is selective hardening—applying stronger protection only where it is most needed.

8. CONCLUSIONS

This study presents a secure search framework that combines traditional keyword matching with the semantic understanding of BERT through a dual-index system. It shows that encrypted data can be searched without compromising security or contextual understanding. This feature could benefit healthcare professionals searching for patient histories, finance teams scrutinizing transactions. Also, attorneys seeking legal documents even when alternative terms are employed. It reduces query duration by 40% and enhances both precision and productivity. In addition, facilitating quicker decryption via LFEHI, while also accommodating fuzzy matching. The use of se-mantic expansion results in a 22% increase in recall, reaching nearly 90% accuracy. Also, SGX guarantees targeted and secure decryption with minimal memory requirements. The system achieves a query time of 64 milliseconds, and offers 94.6% tolerance for errors. It also has 18.5% storage overhead. Nonetheless, the tested scalability is restricted to 1,000 documents, there are no advanced side-channel protections, BERT introduces additional computational expenses, and presently, the model functions solely in single-user settings without features for regulatory compli-ance.

8.1 Future Research Directions

i. Differential privacy: Future advancements may focus on the field of privacy-preserving ana-lytics. This would permit organizations to examine encrypted search logs without the need to access the specific inquiries made by individual users. The system is capable of adding controlled noise to shield sensitive inquiries—either at the user's device level or in a secure computing environment. By doing so, the integrity of individual behaviors is retained. While organizations persist in deriving meaningful overarching trends and insights.

ii. Multimodal search: Subsequent efforts will focus on facilitating encrypted searches across vis-uals, spreadsheets, and graphs through vision-language models like CLIP. The querying of docu-ments that combine text and visuals is enabled by enhancement. Which is beneficial for sectors in-cluding healthcare, finance, and legal studies. Nevertheless, it introduces obstacles pertaining to storage efficiency, similarity assessment. Also, the secure encryption of visual materials.

iii. Blockchain verification: Future systems might use blockchain and zero-knowledge proofs to-gether. The search results are accurate and complete by checking the users. They don’t have to just trust the server. Making encrypted searches transparent and auditable by smart contracts would store verifiable proofs.

REFERENCES

[1] C. Xu, R. Wang, L. Zhu, C. Zhang, R. Lu, and K. Sharif, "Efficient strong privacy-preserving conjunctive keyword search over encrypted cloud data," IEEE Transactions on Big Data, vol. 9, no. 3, pp. 805-817, 2023. https://doi.org/10.48550/arXiv.2203.13662.

[2] Y. Liang, Y. Li, K. Zhang, and Z. Wu, "VMSE: Verifiable multi-keyword searchable encryption in multi-user setting supporting keywords updating," Journal of Information Security and Applications, vol. 76, p. 103518, 2023. https://doi.org/10.1016/j.jisa.2023.103518.

[3] Y. F. Tseng, C. I. Fan, and Z. C. Liu, "Fast keyword search over encrypted data with short ciphertext in clouds," Journal of Information Security and Applications, vol. 70, p. 103320, 2022. https://doi.org/10.1016/j.jisa.2022.103320.

[4] H. Yin, Y. Li, H. Deng, W. Zhang, Z. Qin, and K. Li, "Practical and dynamic attribute-based keyword search supporting numeric comparisons over encrypted cloud data," IEEE Transactions on Services Computing, vol. 16, no. 4, pp. 2855-2867, 2023. doi: 10.1109/TSC.2022.3225112.

[5] K. He, J. Guo, J. Weng, J. K. Liu, and X. Yi, "Attribute-based hybrid boolean keyword search over outsourced encrypted data," IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 6, pp. 1207-1217, 2020. doi: 10.1109/TDSC.2018.2864186.

[6] N. H. Sultan, N. Kaaniche, M. Laurent, and F. A. Barbhuiya, "Authorized keyword search over outsourced encrypted data in cloud environment," IEEE Transactions on Cloud Computing, vol. 10, no. 1, pp. 216-233, 2022. DOI: 10.48175/IJARSCT-18468.

[7] J. Chen, X. Zhang, Y. Li, R. Yang, Y. Liu, and B. Xu, "EliMFS: Achieving efficient leakage-resilient and multi-keyword fuzzy search on encrypted cloud data," IEEE Transactions on Services Computing, vol. 13, no. 6, pp. 1072-1085, 2020. https://doi.org/10.1109/TSC.2017.2765323.

[8] X. Li, Y. Liu, J. Zhang, and M. Chen, "VRFMS: Verifiable ranked fuzzy multi-keyword search over encrypted data," IEEE Transactions on Services Computing, vol. 16, no. 1, pp. 698-710, 2023. doi:10.1109/TSC.2021.3140092.

[9] Q. Tong, Y. Miao, J. Weng, X. Liu, K. K. R. Choo, and R. H. Deng, "Verifiable fuzzy multi-keyword search over encrypted data with adaptive security," IEEE Transactions on Knowledge and Data Engineering, 2023. doi: 10.1109/TKDE.2022.3152033.

[10] H. Shen, L. Xue, H. Wang, L. Zhang, and J. Zhang, "B+-tree based multi-keyword ranked similarity search scheme over encrypted cloud data," IEEE Access, vol. 9, pp. 150865-150877, 2021. doi: 10.1109/ACCESS.2021.3125729.

[11] B. He and T. Feng, "Encryption scheme of verifiable search based on blockchain in cloud environment," Cryptography, vol. 7, no. 2, p. 16, 2023. https://doi.org/10.3390/cryptography7020016.

[12] J. Du, J. Zhou, Y. Lin, W. Zhang, and J. Wei, "Secure and verifiable keyword search in multiple clouds," IEEE Systems Journal, vol. 16, no. 2, pp. 2660-2671, 2022. doi: 10.1109/JSYST.2021.3069200.

[13] H. Dai, M. Yang, G. Yang, Y. Xiang, Z. Hu, and H. Wang, "A keyword-grouping inverted index based multi-keyword ranked search scheme over encrypted cloud data," IEEE Transactions on Sustainable Computing, vol. 7, no. 3, pp. 561-578, 2022. doi: 10.1109/TSUSC.2021.3125520.

[14] S. A. Jabber, S. Hashem, and S. Jafer, "Secure cloud computing by a dual-layer encryption mechanism," Preprints, 2023. https://doi.org/10.20944/preprints202312.0615.v1.

[15] D. Shivaramakrishna and M. Nagaratna, "A novel hybrid cryptographic framework for secure data storage in cloud computing: Integrating AES-OTP and RSA with adaptive key management and time-limited access control," Alexandria Engineering Journal, vol. 84, pp. 275-284, 2023. https://doi.org/10.1016/j.aej.2023.10.054.

[16] H. Tariq and P. Agarwal, "Secure keyword search using dual encryption in cloud computing," International Journal of Information Technology, vol. 12, no. 4, pp. 1063-1072, 2018. https://doi.org/10.1007/s41870-018-0091-6.

[17] H. Yi, "Improving cloud storage and privacy security for digital twin based medical records," Journal of Cloud Computing: Advances, Systems and Applications, vol. 12, no. 1, 2023. https://doi.org/10.1186/s13677-023-00523-6.

[18] Q. Zhang, G. Wang, and Q. Liu, "Enabling cooperative privacy-preserving personalized search in cloud environments," Information Sciences, vol. 480, pp. 1-13, 2019. https://doi.org/10.1016/j.ins.2018.12.016.

[19] N. S. J. Sambrekar and K. P. S., "Privacy-preserving and ranked search using advanced multi-keyword scheme over the encrypted cloud environment," Journal of Electrical Systems, vol. 20, no. 1s, pp. 353-365, 2024. DOI: https://doi.org/10.52783/jes.776.

[20] S. Memon, A. Lakhan, and Q. U. A. Mastoi, "AQ-ResCon: Adaptive quantum-resistant lattice-based key agreement protocol for secure distributed container orchestration in edge cloud environments," International Journal of Mathematics, Statistics, and Computer Science, vol. 3, pp. 377-389, 2025. DOI: https://doi.org/10.59543/ijmscs.v3i.15091.

[21] A. Lakhan, M. Mohammed, L. Al-Budair, S. Memon, V. Slaný, M. Deveci, and R. Martinek, "Enhancing transparency and efficiency in blockchain harvest: Empowering farmers and consumers through transparent trading in agricultural applications," Alexandria Engineering Journal, vol. 118, 2025. https://doi.org/10.1016/j.aej.2025.01.005.

[22] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 380-388, 2002. https://doi.org/10.1145/509907.509965.

[23] D. J. Bernstein, "ChaCha, a variant of Salsa20," in Workshop Record of SASC 2008: The State of the Art of Stream Ciphers, pp. 1-6, 2008. https://cr.yp.to/chacha/chacha-20080120.pdf.

[24] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in Advances in Cryptology-CRYPTO '96, Lecture Notes in Computer Science, vol. 1109, pp. 1-15, 1996. https://dl.acm.org/doi/10.5555/646761.706031.

[25] S. E. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," Foundations and Trends in Information Retrieval, vol. 3, no. 4, pp. 333-389, 2009. https://doi.org/10.1561/1500000019.

[26] C. D. Manning, P. Raghavan, and H. Schütze, "Introduction to Information Retrieval". Cambridge, U.K.: Cambridge University Press, 2008. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.