

LEXICURE: Efficient, Encrypted Lexical Search

Malvika Raj Joshi
UC Berkeley*

Tal Wagner
AWS and Tel Aviv University

Shai Halevi
AWS

Nina Mishra
AWS

Abstract

Lexical search retrieves documents containing query keywords that require exact matches, such as rare terms, entity names or codewords. Search is increasingly hosted in the cloud, which places both the corpus and every query on infrastructure the data owner does not control. Encrypted lexical search addresses this concern by ensuring no third party, including the cloud provider, learns which queries are issued or which documents are retrieved. The key binding constraints are all on the client — accuracy, memory, computation, and communication — since cloud capacity is elastic. Existing approaches that guarantee full privacy without caveats require prohibitively high client-side persistent storage or communication.

We present LEXICURE, a protocol for encrypted lexical search that reduces the search to arithmetic operations favorable to cryptographic primitives, such as encrypted matrix–vector products (EMVP). Across sixteen diverse datasets, LEXICURE retains over 95% of plaintext BM25 accuracy on average. LEXICURE sets a new baseline for leakage-free lexical search, requiring $2.6\text{--}9.9\times$ less communication than alternative approaches and less than 1KB of client-side storage. A major challenge is that secure lexical search is inherently bandwidth-bound. We additionally describe an approach for compressing the communication further in exchange for a larger server. To do so, we introduce a new cryptographic primitive that modifies EMVP to securely compute quadratic and higher-degree functions, more efficiently than a direct reduction.

LEXICURE demonstrates that leakage-free encrypted search with practical performance is achievable and opens new avenues for secure information retrieval systems.

*Work done while at AWS.

Contents

1	Introduction	4
1.1	Lexical, Semantic, and Hybrid Search	4
1.2	Our Contributions	5
1.3	Technical Overview	6
1.3.1	Lexical Scoring as Matrix-Vector Product	6
1.3.2	Architecture Choice: Using EMVP	6
1.3.3	The LEXICURE Protocols	7
1.3.4	Extending EMVP Beyond Linear Functions	8
1.4	Related Work	8
1.5	Organization	8
2	System Model and Goals	9
2.1	Threat Model	9
2.2	Accuracy and Efficiency	9
3	The Lexicure-1 Protocol	10
3.1	Lexical Search and BM25 Ranking	11
3.2	Reducing the Dimension with CountSketch	11
3.3	Encrypted Matrix-Vector Product (EMVP)	12
3.3.1	The EMVP Protocol from [4]	12
3.4	LEXICURE-1: One-Round Protocol	13
3.4.1	The bandwidth bottleneck	14
4	Higher-Degree EMVP	15
4.1	EMVP ² : Encrypted Bilinear Forms	15
4.1.1	Optimizations for quadratic forms	16
4.2	Use Cases for Higher-Degree Functions	17
5	Lexicure-2: Two-Round Protocol with Sublinear Communication	18
5.1	Document Partitioning	18
5.2	Round One: Group-Score Aggregation	19
5.3	Round Two: Private Group Retrieval	19
5.4	Batching TDM Evaluation	20
5.5	Setup Complexity	21
6	Evaluation	21
6.1	Experimental Setup	22
6.1.1	Datasets	22
6.1.2	Evaluation Metrics	22
6.2	Comparison to existing approaches	23
6.2.1	Results	24
6.3	Additional evaluations of LEXICURE	26
6.3.1	LEXICURE-2: Communication improvements	27

7	Conclusion	28
7.1	Combining Lexical and Semantic Search	29
7.2	Beyond Semi-Honest Security	29
7.3	Dynamic Datasets	29
7.4	Hybrid EMVP-FHE Approaches	30
7.5	Preprocessing and Clustering	30
7.6	Parameter Optimization	31
7.7	Document Retrieval	31
8	Acknowledgements	31
A	Details on BM25 Scoring	40
B	Formal Metric Definitions	41
C	Cryptographic Definitions and Proofs	42
C.1	The 1D-SLSN Assumption	43
C.2	Security Proofs	43
D	Additional Evaluation Details	44
D.1	Implementation Details	44
D.2	Sources of Accuracy Loss	45
D.3	Additional LEXICURE-1 Results	45
D.3.1	Measured Client Computation	45
D.3.2	Projected Server Computation	46
D.4	Accuracy at a Rank Cutoff of 100	46
D.5	Additional LEXICURE-2 Results	47
D.5.1	Additional Accuracy Metrics	47
D.5.2	Bandwidth and Server Storage	47
D.5.3	Client and Server Computation	48
D.5.4	Locality Analysis	49
E	Alternative Sketching Methods	49

1 Introduction

Documents are stored in the cloud so that they can be found again. Encrypting the documents to hide their contents from the cloud provider appears to be in direct conflict with this search. Although a cloud can store ciphertexts, conventional cloud-search systems cannot search or rank them without decrypting. The owner of a confidential corpus is faced with an impossible choice — store the data on premises and forgo cloud-scale search, or upload it in the clear and extend trust to the provider’s software, its operations, and anyone who breaches them. Patient records, legal discovery sets, confidential meeting transcripts, and source material a journalist has promised to protect all have to be searched, yet none of them can be handed over. Some designs try to narrow that trust requirement to “just” trusted hardware in the cloud [83] (e.g. Intel SGX [26]), or split the server among multiple parties trusted not to collude [23]. Neither assumption is innocuous. Enclave keys have been extracted by transient-execution attacks [13], and if the parties ever collude, no security remains.

Encrypted search promises a way out: the client holds the keys, the server holds only ciphertexts, and search still works. Efforts to enable encrypted search go back at least a quarter century (cf. Song et al. [87, 38, 27]), and it is worth exploring why such systems are still not widely used. Three properties are wanted at once: (i) reveal nothing to the server, (ii) retrieve as well as plaintext search, and (iii) cost little enough to run. Nearly all existing systems give up one of these aspects, and the sacrifice is structural rather than incidental.

Most works in the literature give up the first: Searchable symmetric encryption [87, 38, 27, 19, 53, 90, 16, 8, 9] buys its speed by letting the server observe which encrypted entries a query touches, how often, and how many results come back. This residual signal, access-pattern leakage, turns out to be a real problem. Although the server “merely” sees which ciphertexts match, a sustained line of attacks shows that this suffices to expose the queries and much of the database in the clear [60, 51, 15, 72, 78, 65, 30, 6, 48, 76, 56].

The lines of work that eliminate this leakage instead give up on cost, in two complementary ways. One line of solutions uses Oblivious RAM [39] to hide the query-dependent memory accesses [89, 81]. An inverted index over ORAM yields leakage-free keyword retrieval, but it treats the server as dumb memory, leaving the search itself to the client. The client must perform all computations locally, on top of absorbing the ORAM overhead of every memory access, ultimately negating most of the cloud’s benefit. Another line uses Fully Homomorphic Encryption [37, 1] and PIR [23, 58], and it has the opposite problem. The client is lighter than in ORAM-based solutions, but general-purpose homomorphic evaluation can make the server infeasible, keeping such systems out of production [68].

The challenge of providing leakage-free search that keeps *both* endpoints within budget, a client light enough to run on a phone or laptop and a server whose computation is not prohibitively expensive, remains.

1.1 Lexical, Semantic, and Hybrid Search

Modern search systems support both lexical and semantic search to serve complementary retrieval needs.

Lexical search scores a document by matching terms in the query and weighting them by corpus statistics such as term frequency and inverse document frequency [88]. The workhorse, and the current default in widely deployed search engines such as Lucene, Elasticsearch, and OpenSearch,

is an inverted index [64] ranked using the BM25 scoring function [82]. Lexical scoring is exact, interpretable, and robust on rare terms, entity names, identifiers, and out-of-domain vocabulary, but it cannot capture subtle semantics.

Semantic search uses modern deep learning embedding models to embed documents and queries into a shared dense vector space. Relevance is scored by a distance measure such as cosine similarity [55, 79]. Semantic scoring generalizes across paraphrases, synonyms, and wider semantic context, but it fails to capture long-tail tokens that may carry relevance signal. Take, for instance, an intelligence agent searching through classified records containing tactical code names or obfuscations. In such situations, conceptually related documents do not substitute for those containing the exact keywords. Consequently, the state-of-the-art retrieval systems are *hybrid* and run both lexical and semantic retrieval on the same corpus to merge the results [61]. Hybrid retrieval often outperforms either component alone across heterogeneous benchmarks [20] and is also the recommended default retrieval method for RAG systems [99].

Encrypted semantic search has been addressed by systems such as Tiptoe for public databases [45], and Compass [101] and BiSON [31] for encrypted private databases. These advances address only one half of a typical hybrid search system, while leakage-free encrypted lexical search remains largely unexplored. The challenge, once again, is structural. Plaintext approaches such as inverted indexes derive their efficiency from term-frequency patterns in natural-language data. These are precisely the statistics that leakage-free search must hide; moreover, layering on primitives like ORAM can end up worse than downloading the entire index to the client [70, 5].

This work lays the groundwork for making leakage-free lexical search a reality, and sets a new baseline for its cost.

1.2 Our Contributions

We present LEXICURE, the first leakage-free encrypted lexical-search system with an efficient client and a feasible server. Our setting is the natural one, where a client outsources an encrypted document collection to a powerful, untrusted server, then issues lexical queries against it. The (semi-honest) server learns nothing beyond public parameters such as corpus size and the protocol used. In particular, the server does not learn the contents of the documents, nor the queries, nor whether two queries are equal or return the same answers. Following standard practice in this literature [101, 27], a query returns a ranked list of document identifiers with relevance scores, and content retrieval is a separate step performed using PIR [23, 58]. As a starting point, we focus on static indexes, and discuss extensions to dynamic databases in Section 7.3.

We describe two variants of LEXICURE, both of which operate by reducing the encrypted lexical search to highly-parallelizable encrypted arithmetic. We start with LEXICURE-1, a single-round protocol whose encrypted computations are designed to consist only of linear forms. These computations can be performed at a fraction of FHE’s cost using a recent primitive, EMVP [4]. We evaluate LEXICURE-1 on sixteen datasets, from 3.6K to 8.8M documents, spanning diverse domains to demonstrate that it consistently achieves accuracy close to plaintext BM25 (over 95% on most datasets and metrics). On the largest dataset, MS MARCO, LEXICURE-1 retains 99.8% of the plaintext Recall and over 98% of MRR and NDCG. LEXICURE-1 also sets a new benchmark for communication among leakage-free encrypted lexical-search systems and uses $2.6\text{--}9.9\times$ less bandwidth per query than ORAM-based approaches (Table 7). Since ORAM’s traffic is dominated by uploads, on a typical 5G link, LEXICURE-1’s data transfer is over $50\times$ faster (Fig. 8). LEXICURE-1 is consistently preforms best across different datasets and network settings, saving up to $20\times$ in

network latency compared to the second-best.

LEXICURE, like many secure systems, is bandwidth-bound. Prior to this work, secure lexical search at a scale even remotely in the vicinity of small search engines (billions of pages) was inconceivable. This was despite the availability of major cloud providers such as AWS, Google Cloud, Microsoft Azure, and Oracle Cloud, all offering infrastructure capable of handling hundreds of terabytes of data [2, 40, 57, 24]. As a reference point, the best prior work that we evaluate in Section 6.2 already requires nearly 2 GB of data transfer for a single query on MS MARCO (compared with 0.51 GB in LEXICURE-1).

We offer LEXICURE-2 as a first step across this bandwidth barrier. LEXICURE-2 is capable of aggressively reducing the bandwidth in exchange for more server storage. On MS MARCO, LEXICURE-2 requires $5\times$ less communication than LEXICURE-1, with little-to-no degradation in the retrieval quality. To achieve this, we introduce a *new cryptographic tool* for evaluating encrypted bilinear forms: In Section 1.3.4, we describe a modification of EMVP that extends it to higher-degree functions. When used to compute bilinear forms, the client’s cost remains essentially the same as EMVP. This extension to higher-degree functions may be of independent interest, for example to compute the bilinear forms in attention layers of a transformer architecture [42, 77, 95], see Section 4.2.

1.3 Technical Overview

1.3.1 Lexical Scoring as Matrix-Vector Product

Standard metrics such as BM25 and TF-IDF reduce naturally to an inner-product computation [64, Chapter 6], representing each query as a sparse 0/1 vector over the vocabulary $\mathcal{V}$ and each document as a BM25-weighted vector of the same dimension. The relevance of a document to a query is then their inner product, and ranking the corpus is a matrix-vector product.

However, the dimension of these vectors is as large as $|\mathcal{V}|$, which is too large for encrypted computation systems to handle directly. We use CountSketch to compress these sparse high-dimensional vectors into dense vectors of dimension $d \approx 10^4$ while approximately preserving inner products. These representations form an $N \times d$ document matrix $\mathbf{M}$ and a query vector $\mathbf{q}$, and the relevance scores are then $\mathbf{q}\mathbf{M}^T$. We also discuss alternative sketching methods in Appendix E.

1.3.2 Architecture Choice: Using EMVP

We now have a concrete computational problem: privately evaluate an encrypted matrix-vector product on an $N \times d$ matrix with $d \approx 10^4$. Several cryptographic approaches could serve here, and the choice determines the system’s operating point.

ORAM-based vector search. Systems such as Compass [101] and BiSON [31] use ORAM-protected graph traversal for encrypted (approximate) vector search. This approach has proved effective for semantic search, where embeddings are 128–768 dimensions. However, it targets a fundamentally different operating point: the client drives an iterative graph traversal while retaining auxiliary data and evolving ORAM state, and each query requires multiple rounds of interaction. Moreover, the sketch dimension required to densify BM25 ($d \approx 10^4$) is more than an order of magnitude larger than the semantic embeddings these systems evaluate. This substantially increases ORAM block sizes and communication. Finally, ORAM also requires the server to keep a per-client evolving state, making replication and load-balancing among a fleet of servers very difficult.

FHE-based computation. General-purpose fully homomorphic encryption can evaluate any function on encrypted data, but its computational cost on the server makes it impractical for a matrix-vector product of this scale. For instance, evaluating a bilinear form with FHE requires multiple layers of encrypted multiplication (multiplicative depth at least 2), each involving expensive specialized operations in an extension field. These layers also require auxiliary evaluation keys, relinearization, and noise-management procedures.

EMVP. Recent secure linear algebra protocols [4, 12, 94] offer a restricted but dramatically more efficient alternative to FHE for evaluating linear functions on encrypted data. In Encrypted Matrix-Vector Product [4], the server evaluates an encrypted matrix-vector product by computing a similar matrix-vector product over the same field, but on matrices with somewhat larger dimensions (the same number of rows and typically $1.25\times$ to $4\times$ as many columns). At the same time, the protocol maintains an essentially optimal client whose work is typically linear in the number of rows, as required merely to read the answer. EMVP is not general-purpose, unlike FHE, but it is sufficient for the matrix-vector product primitive we need. We use EMVP to keep the client’s persistent state negligible and the encrypted server-side index unchanged across queries. This choice results in a lightweight client, requiring only the storage of the short encryption keys, while the cost of privacy is absorbed at the server, which requires more storage than ORAM-based systems. For our specific application to lexical search, we find that this choice also leads to much lower communication cost than ORAM-based approaches on typical medium-to-large indexes.

1.3.3 The Lexicure Protocols

In LEXICURE-1, the matrix $\mathbf{M}$ is obtained by densifying the BM25 sparse vectors (using CountSketch). $\mathbf{M}$ is then encrypted according to the EMVP scheme, with the client holding the secret key. During a query, the client uploads an encryption of the densified query $\mathbf{q}$, using its saved keys. The client receives N encrypted scores, decrypts them, and extracts the top- K results. Decryption is linear and top- K selection is incremental, so scores can be processed as they stream in, making the $\Theta(N)$ download a potential bottleneck.

Although LEXICURE-1 already outperforms other potential alternatives to encrypted lexical search, its bandwidth is still linear in the size of the database. For weak clients, processing so much bandwidth could be prohibitive. We therefore also describes LEXICURE-2 that has sublinear communication complexity. LEXICURE-2 partitions an N -document database into $G \ll N$ groups, and has two communication rounds: The server first returns one encrypted aggregate score per group, by computing an encrypted bilinear form of the query. The client picks the top K_G groups (K_G is a parameter that controls the search quality), and privately fetches individual scores from only those groups in round two. Note that the retrieval in the second round is an instance of PIR on a query-dependent encrypted computation result, and is also a bilinear form. With $G \approx \sqrt{K_G N}$, the score download is $O(\sqrt{K_G N})$, providing an adjustable quality-bandwidth tradeoff. This approach performs particularly well when the corpus ordering has locality properties by allowing K_G to be as low as 20 (see Section 6.3.1). Implementing this approach requires an extension of the EMVP protocol, as we now explain.

1.3.4 Extending EMVP Beyond Linear Functions

To identify the top groups in LEXICURE-2, the server needs to compute an aggregate score for each group. Ideally, we would use the largest score in each group, but computing the maximum on encrypted data is expensive. Instead, a natural aggregate is the squared ℓ_2 norm of the group’s encrypted score vector, which is a quadratic form: $\mathbf{q}(\mathbf{M}_i^T \mathbf{M}_i) \mathbf{q}^T$.

The EMVP protocol from [4] computes only linear functions, and using FHE for this quadratic computation would forfeit the efficiency that motivated EMVP in the first place. We instead extend EMVP in a *white-box* manner, i.e., by modifying its internal components to handle bilinear forms at roughly the same cost as the original linear EMVP protocol. More generally, this extension handles degree- δ functions at a cost that grows exponentially with $\delta/2$ rather than δ .

This contribution goes beyond what LEXICURE-2 needs. Plain EMVP is restricted to linear functions, which confines it to a narrow niche despite its efficiency advantage over FHE. Our extension promotes EMVP-based techniques from special-purpose to general-purpose while inheriting many of these advantages (e.g. no relinearization, field-agnostic, does not imply public key encryption). This enables a fast library of privately outsourced low-degree functions that can be composed with FHE for higher-degree stages. See Section 4.2 for further discussion.

1.4 Related Work

Encrypted search has been extensively studied under various security models and retrieval goals. Many encrypted lexical-search systems use searchable symmetric encryption (SSE) [14]. SSE achieves high performance by revealing structured information such as search and access patterns, which violates our threat model [87, 27]. Indeed, these patterns have enabled query-recovery and database-reconstruction attacks; see the survey [60] and references therein.

In Section 6.2, we compare LEXICURE against other existing encrypted lexical search baselines that operate under the same threat model as ours. These are described in Compass [101] and consist of an oblivious inverted index (Inv-ORAM, and our extension Inv-O-BM25), and the application of the clustering approach in Tiptoe [45] to private databases (HE-Cluster). The leakage-free alternatives beyond these generally rely on trusted hardware or non-collusion (e.g. [67, 32]). The structural challenges of preventing leakage is also present in these settings; see Obliv [67] and POSUP [46] (not fully leakage-free). Early works on SSE [70, 5] caution that addressing leaking through ORAM can cost more than a straightforward leakage-free download. Our experiments in Fig. 8 confirm these concerns.

For *semantic* search, two notable works that address the same threat model as ours are Compass [101] and its successor, BiSON [31]. Both apply ORAM-protected graph indexes to semantic search. BiSON substantially improves the scalability and performance of this approach using DiskANN [52] and a disk-compatible Ring ORAM, but retains the large client-side persistent storage [31]. Although a recursive ORAM can reduce client-side persistent storage, it also increases communication, whereas, both these systems reduce communication through persistent client-side storage.

1.5 Organization

Section 2 describes our system model, security definition, and design goals. Section 3 presents the single-round LEXICURE protocol. Section 4 then describes the EMVP² construction and its extension to higher-degree functions. Section 5 uses EMVP² to construct the two-round LEXICURE protocol. Section 6 evaluates both protocols on sixteen datasets, compares them with recent

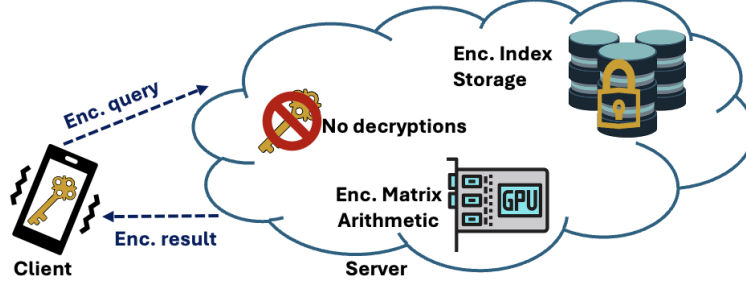


Figure 1: Encrypted-search model: a weak client with a secret key interacts with a powerful, untrusted server.

baselines, and analyzes their accuracy, bandwidth, and storage tradeoffs. Section 7 concludes with limitations and directions for future work.

2 System Model and Goals

Our system model is sketched in Fig. 1. We consider a weak client that interacts with an untrusted server providing storage and computation services. The client owns a static dataset and wants to store it on the server while retaining the ability to search it. Specifically, the data consists of sensitive text documents, and the client needs to perform lexical search on these documents while keeping both data and queries secret from the server.

In a one-time initialization step, the client encrypts the document collection and uploads it to the server, retaining only the secret keys. Following standard practice in searchable encryption [101, 27], we treat index construction as an offline setup phase and focus on optimizing the online query phase. The goal is to support text-based lexical search. Queries consist of terms from a fixed vocabulary $\mathcal{V}$, and the operation returns a ranked list of K document IDs with their relevance scores based on term matching between the query and documents.

2.1 Threat Model

We assume a *semi-honest* server that follows the prescribed protocol but should be kept from learning anything about the data or queries or about the client’s access pattern. (For example, the server should not learn if two queries are the same, or if they return the same answer.) The game-based security experiment is described in Fig. 2.

Definition 2.1. *A two-party encrypted-search protocol ensures semantic security for the data and queries against a semi-honest server if, for every polynomial-time adversary $\mathcal{A}$ in the game from Fig. 2, there is a function negligible in the security parameter such that $\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + \text{negligible}(\lambda)$.*

2.2 Accuracy and Efficiency

Together with the security goal above, we also need the protocol to maintain the accuracy of the results and keep a lightweight client and a practical server.

1. The challenger $\mathcal{C}$ chooses a size- λ secret key and a random $b \in_{\$} \{0, 1\}$.
2. The adversary $\mathcal{A}$ chooses two equally large datasets D_0, D_1 .
3. $\mathcal{C}$ encrypts D_b and sends the encrypted dataset to $\mathcal{A}$.
4. $\mathcal{A}$ repeats polynomially many times, in each step i :
 - (a) $\mathcal{A}$ chooses a pair of queries $q_{i,0}, q_{i,1}$.
 - (b) $\mathcal{C}$ encrypts $q_{i,b}$ and sends the encrypted query to $\mathcal{A}$.
 - (c) $\mathcal{C}$ and $\mathcal{A}$ interact according to the protocol on encrypted dataset D_b and query $q_{i,b}$.
5. $\mathcal{A}$ outputs a guess b' and wins the execution if $b' = b$.

Figure 2: Security game for secret lexical search. The protocol is secure if for any feasible adversary $\mathcal{A}$, $\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + \text{negligible}(\lambda)$.

Accuracy. We aim to match the search quality of plaintext BM25 ranking. We evaluate our protocol using standard information-retrieval metrics (MRR@10, NDCG@10, and Recall@10) on 16 benchmark datasets from BEIR [91] and TripClick [80], spanning diverse domains and corpus sizes.

Lightweight client. We obtain an essentially stateless client that stores only short keys, and our protocols use a constant number of communication rounds. This contrasts with ORAM-based approaches that require significant client-side storage (hundreds of MB) for efficiency [101]. Beyond the asymptotics, we also analyze the concrete cost of our protocol on common benchmark datasets and compare it with recent approaches. Our evaluation shows that the protocol achieves lower bandwidth than existing solutions at comparable accuracy levels.

Practical server. In our setting the server provides both storage and computation. This mirrors real-world search systems, in which clients are typically personal devices while search services have data-center-scale resources. While allowing computational overhead for security, the server must remain practical. We extend the recent EMVP protocol [4], which provides significant efficiency gains over FHE, to obtain a new primitive for computing quadratic forms. Using this primitive, the server’s computation can all be built from dense matrix/vector arithmetic, just as in the EMVP protocol. These operations are highly optimized and can be executed efficiently on modern hardware.

3 The Lexicure-1 Protocol

We describe our single-round leakage-free encrypted search protocol, which achieves high accuracy with a lightweight client through arithmetization of the search process. As previously discussed, lexical relevance scoring can be cast naturally as a matrix-vector product (MVP) over sparse, very-high-dimensional vectors. We use sketching to compress these vectors into dense vectors of a lower, practical dimension and then invoke the EMVP protocol from [4] to achieve efficient, leakage-free lexical search.

We first describe the main components: BM25 ranking, CountSketch compression, and the EMVP protocol, and then combine them into a single-round protocol in Section 3.4 as illustrated in Fig. 3. The main drawback is that the client has to download $O(N)$ encrypted field elements

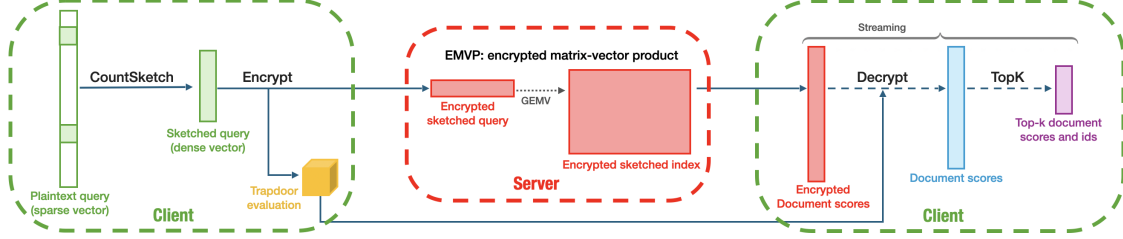


Figure 3: LEXICURE-1 query pipeline. The final document scores are an approximation of BM25 scores (or other relevance metric). The top-k and their corresponding IDs (row index) are recovered, the corresponding document contents are retrieved in a separate step from the server with standard PIR.

from the server (Section 3.4.1), where N is the number of documents in the corpus. In Section 5 we describe a two-round variant, LEXICURE-2, that addresses this drawback.

3.1 Lexical Search and BM25 Ranking

Lexical search operates over a fixed vocabulary $\mathcal{V}$ of searchable terms. Let $\mathcal{D}$ be a corpus of documents over $\mathcal{V}$, where $V = |\mathcal{V}|$. A lexical query $Q \subseteq \mathcal{V}$ consists of a set of terms, and documents are ranked according to their relevance to these terms. For ease of exposition and in our evaluations, we use the BM25 ranking function [82], a standard and widely used lexical scoring function. Nonetheless, the approach here is adaptable to most lexical-scoring functions. The relevance of a query Q and document $D \in \mathcal{D}$ is given by,

$$\text{rel}_{\mathcal{D}}(Q, D) = \sum_{q \in Q} \text{tf}_{\mathcal{D}}(q, D) \cdot \text{idf}_{\mathcal{D}}(q), \quad (3.1)$$

where $\text{tf}_{\mathcal{D}}(q, D)$ captures the prevalence of q in D , while $\text{idf}_{\mathcal{D}}(q)$ penalizes terms that appear in many documents. These quantities additionally depend on corpus-wide and global hyperparameters, as detailed in Appendix A. Q and D can equivalently be encoded into a sparse vector representation as bag-of-words vectors $\mathbf{v}_D, \mathbf{u}_Q \in \mathbb{R}^V$. The entries in $\mathbf{v}_D$ are weighted by their respective TF-IDF values, while $\mathbf{u}_Q$ is unweighted and supported only on the query terms. With this encoding, the relevance can be computed as the inner product $\text{rel}_{\mathcal{D}}(Q, D) = \langle \mathbf{u}_Q, \mathbf{v}_D \rangle$.

Once the corpus and BM25 parameters are fixed, every document vector $\mathbf{v}_D$ can be computed in advance; only the sparse query vector $\mathbf{u}_Q$ is constructed at search time.

3.2 Reducing the Dimension with CountSketch

As noted before, the vocabulary size V is typically very large (often 10^5 to 10^6 terms), making direct EMVP computation on V -dimensional vectors impractical, and even more so due to the large upload. We therefore use sketching as a “densifier” that maps the high-dimensional sparse vectors $\mathbf{u}_Q$ and $\mathbf{v}_D$ to lower-dimensional dense vectors while approximately preserving their inner products. LEXICURE uses the CountSketch densifier [18] because it is efficient to compute and yields good search quality. (See Appendix E for other hashing methods that we considered.)

CountSketch uses two hash functions: a pairwise independent hash $h : [V] \rightarrow [d]$ and a 4-wise independent hash $g : [V] \rightarrow \{-1, 1\}$, where $d \ll V$ is the target dimension. For a sparse vector

$\mathbf{a} \in \mathbb{R}^V$, the corresponding dense vector $\mathbf{v}(\mathbf{a}) \in \mathbb{R}^d$ is defined by

$$\mathbf{v}(\mathbf{a})_j := \sum_{i \in h^{-1}(j)} a_i \cdot g(i). \quad (3.2)$$

It is easy to see that $\langle \mathbf{v}(\mathbf{a}), \mathbf{v}(\mathbf{b}) \rangle = \langle \mathbf{a}, \mathbf{b} \rangle$ in expectation. Furthermore, we have the following tail bound for CountSketch [93, 54], which holds for each fixed pair of vectors $\mathbf{a}, \mathbf{b}$:

$$\forall \delta \in (0, \frac{1}{2}), \Pr_{h,g} [|\langle \mathbf{v}(\mathbf{a}), \mathbf{v}(\mathbf{b}) \rangle - \langle \mathbf{a}, \mathbf{b} \rangle| > \frac{3}{\sqrt{\delta d}} \|\mathbf{a}\| \|\mathbf{b}\|] < \delta. \quad (3.3)$$

By increasing the sketch dimension d , we can make the approximation error arbitrarily small. In our implementation, we uniformly fix $d = 10^4$ for every dataset. We determined this value experimentally to yield good accuracy while maintaining practical performance.

3.3 Encrypted Matrix-Vector Product (EMVP)

EMVP is a protocol for privately outsourcing the computation of linear functions. As defined in [4, Def 2.1], it consists of five procedures:

- $\text{sk} \leftarrow \text{EMVP.KEYGEN}(\$)$, run by the client to generate a secret key;
- $\hat{\mathbf{M}} \leftarrow \text{EMVP.ENCRYPTMATRIX}(\text{sk}, \mathbf{M})$, run by the client to encrypt a matrix representing the linear function, $\hat{\mathbf{M}}$ stored at the server;
- $(\hat{\mathbf{q}}, \tau) \leftarrow \text{EMVP.ENCRYPTQUERY}(\text{sk}, \mathbf{q})$, run by the client to encrypt a query vector, $\hat{\mathbf{q}}$ sent to the server and the state τ kept for decryption;
- $\mathbf{A} \leftarrow \text{EMVP.EVALUATE}(\hat{\mathbf{M}}, \hat{\mathbf{q}})$, run by the server. The encrypted answer $\mathbf{A}$ is returned to the client;
- $\mathbf{a} \leftarrow \text{EMVP.DECRYPT}(\text{sk}, \tau, \mathbf{A})$, run by the client to recover the plaintext answer.

Correctness means that the client obtains $\mathbf{a} = \mathbf{q}\mathbf{M}^T$. Security means that even if the server could choose the matrix and queries itself, it still couldn't distinguish their encryptions from encryptions of random matrices and queries with the same dimensions.

3.3.1 The EMVP Protocol from [4]

The protocol from [4] multiplies matrices over finite fields/rings. We use it with a matrix $\mathbf{M} \in \mathbb{F}_p^{N \times d}$ and vectors $\mathbf{q} \in \mathbb{F}_p^d$, where $p \approx 2^{30}$ is prime. Other parameters in that protocol are the length $n > d$ and rank $k = n - d$ of an $[n, k]$ linear error-correcting code, a *block-length* parameter b , and the number of blocks $s = \lceil n/b \rceil$ (also called the *split parameter*). For our purposes, we use the setting from [4, Table 4]: $d = 10000$, $b = 2728$, $k = 30000$, and $n = 40000$ (which implies the split parameter $s = \lceil 40000/2728 \rceil = 15$).

Key generation. The client draws a fresh random secret key and uses it to derive a random $[n, k]$ -linear code over $\mathbb{F}_p$ from some coding family with fast encoding for both the primal and dual code. Let $\mathbf{C} \in \mathbb{F}_p^{k \times n}$ be a generator matrix for C and $\mathbf{D} \in \mathbb{F}_p^{d \times n}$ a generator matrix for its orthogonal code (so $\mathbf{D}\mathbf{C}^T = \mathbf{0}^{d \times k}$), and assume that $\mathbf{D}$ is systematic, $\mathbf{D} = [\mathbf{I}|\mathbf{D}']$. The client also derives

from the secret key a trapdoored pseudorandom matrix $\mathbf{Z} \in \mathbb{F}_p^{N \times n}$, where the trapdoor enables fast multiplication by $\mathbf{Z}$ of arbitrary n -vectors over $\mathbb{F}_p$ [12, 94].

Matrix encryption. The client encrypts a matrix $\mathbf{M} \in \mathbb{F}_p^{N \times d}$ as $\hat{\mathbf{M}} = \mathbf{M}\mathbf{D} + \mathbf{Z} \in \mathbb{F}_p^{N \times n}$.

Query encryption. The client encrypts $\mathbf{q} \in \mathbb{F}_p^d$ as follows:

- Mask the query with a random codeword: Choose $\mathbf{r} \leftarrow_{\$} \mathbb{F}_p^k$ and set $\tilde{\mathbf{q}} = (\mathbf{q} \parallel 0^k) + \mathbf{r}\mathbf{C} \in \mathbb{F}_p^n$;
- Break $\tilde{\mathbf{q}}$ into s size- b blocks $(\tilde{\mathbf{q}}_1 \parallel \dots \parallel \tilde{\mathbf{q}}_s)$, choose s random invertible scalars $\alpha_1, \dots, \alpha_s \leftarrow_{\$} \mathbb{F}_p^*$;
- Send $\hat{\mathbf{q}} = (\alpha_1 \cdot \tilde{\mathbf{q}}_1 \parallel \dots \parallel \alpha_s \cdot \tilde{\mathbf{q}}_s)$ to the server.

The client also uses the trapdoor for $\mathbf{Z}$ to calculate $\mathbf{r}' = \tilde{\mathbf{q}}\mathbf{Z}^T$ quickly, and saves $\mathbf{r}'$ and the α_i values for decryption. Benhamouda et al. formulate a new hardness assumption, called 1D-SLSN, that underlies query security [4].

Encrypted matrix-vector product. Upon receiving the encrypted matrix $\hat{\mathbf{M}}$, the server breaks it into s blocks $\hat{\mathbf{M}} = [\hat{\mathbf{M}}_1 \parallel \dots \parallel \hat{\mathbf{M}}_s]$, each with b columns. Upon receiving the encrypted query $\hat{\mathbf{q}} = (\hat{\mathbf{q}}_1 \parallel \dots \parallel \hat{\mathbf{q}}_s)$, the server computes $\mathbf{a}'_i = \hat{\mathbf{q}}_i \hat{\mathbf{M}}_i^T \in \mathbb{F}_p^N$ and returns $\mathbf{A} = (\mathbf{a}'_1 \parallel \dots \parallel \mathbf{a}'_s) \in \mathbb{F}_p^{N \times s}$.

Decrypting the answer. The client computes $\mathbf{a}_i = \alpha_i^{-1} \cdot \mathbf{a}'_i$, and decrypts the result as $\mathbf{a} = (\sum_{i=1}^s \mathbf{a}_i) - \mathbf{r}'$. (It is a straightforward algebraic manipulation to show that $\mathbf{a} = \mathbf{q} \cdot \mathbf{M}^T \in \mathbb{F}_p^N$.)

Complexity. The server's work is a matrix-vector product over the same field, using an $N \times n$ matrix rather than the $N \times d$ plaintext matrix. Most of the client's online work is the computation of the product $\mathbf{r}' = \tilde{\mathbf{q}}\mathbf{Z}^T$, which (a) can be done quickly using the trapdoor for $\mathbf{Z}$, and (b) can be performed while it is waiting for the answer from the server. Once the answer arrives from the server, decryption takes essentially sN time and is embarrassingly parallelizable.

3.4 Lexicure-1: One-Round Protocol

LEXICURE-1 is the natural combination of the ingredients from above, see a pseudocode in Algorithm 1 and Fig. 3. Given an N -document input corpus and the search vocabulary, the client first draws a random CountSketch seed σ and generates an EMVP secret key $\mathbf{sk}$. The client computes the sparse BM25 vector for each document, compresses it to dimension $d = 10^4$ using CountSketch with seed σ , and interprets the output as a vector over the field $\mathbb{F}_p$, where $p \approx 2^{30}$. Placing all compressed vectors in a matrix $\mathbf{M} \in \mathbb{F}_p^{N \times d}$, the client encrypts it using EMVP and stores the encrypted $\hat{\mathbf{M}}$ at the server. The client is essentially stateless, storing only the short seed to generate the CountSketch hash functions for query encoding and the secret key for encryption and decryption.

To issue a query $Q \subset \mathcal{V}$, the client encodes it as a high-dimensional sparse 0/1 vector, compresses it with CountSketch to obtain $\mathbf{q} \in \mathbb{F}_p^d$, encrypts it using EMVP, and sends the ciphertext to the server. The server performs the encrypted computation and returns an $N \times s$ encrypted answer matrix, which the client decrypts to obtain the similarity vector $\mathbf{r} = \mathbf{q}\mathbf{M}^T \in \mathbb{F}_p^N$. Finally, the client finds the top K scores in $\mathbf{r}$, corresponding to the documents to retrieve. The total online bandwidth and server storage for LEXICURE-1 are

$$\text{Upload} = n, \text{ Download} = Ns, \text{ Server storage} = Nn \quad (3.4)$$

field elements.

Algorithm 1 LEXICURE-1: 1-round encrypted lexical search

Offline setup (parameter: prime $p \approx 2^{30}$):**Client Input:** Vocabulary $\mathcal{V}$, Documents $(D_1, \dots, D_N)$

- 1: Sample a CountSketch seed σ defining $h : [V] \rightarrow [d]$ and $g : [V] \rightarrow \{-1, 1\}$
- 2: **for** $j \leftarrow 1$ to N **do**
- 3: $\mathbf{v}_j \leftarrow \text{BM25VECTOR}(D_j)$
- 4: $\mathbf{M}_{j,*} \leftarrow \text{COUNTSKETCH}(\mathbf{v}_j, \sigma) \in \mathbb{F}_p^d$
- 5: **end for**
- 6: $\mathbf{sk} \leftarrow \text{EMVP.KEYGEN}$
- 7: $\hat{\mathbf{M}} \leftarrow \text{EMVP.ENCRYPTMATRIX}(\mathbf{sk}, \mathbf{M})$
- 8: Send $\hat{\mathbf{M}}$ to the server; retain $(\mathbf{sk}, \sigma)$

Online query**Client Input:** Query $Q \subseteq \mathcal{V}$ and result count K *Client:*

- 9: $\mathbf{q} \leftarrow \text{COUNTSKETCH}(\mathbf{1}_Q, \sigma) \in \mathbb{F}_p^d$
- 10: $(\hat{\mathbf{q}}, \tau_{\mathbf{q}}) \leftarrow \text{EMVP.ENCRYPTQUERY}(\mathbf{sk}, \mathbf{q})$
- 11: Send $\hat{\mathbf{q}}$ to the server

Server:

- 12: $\mathbf{A} \leftarrow \text{EMVP.EVALUATE}(\hat{\mathbf{M}}, \hat{\mathbf{q}})$
- 13: Send $\mathbf{A} \in \mathbb{F}_p^{N \times s}$ to the client

Client:

- 14: $\mathbf{r} \leftarrow \text{EMVP.DECRYPT}(\mathbf{sk}, \tau_{\mathbf{q}}, \mathbf{A}) = \mathbf{q}\mathbf{M}^T$
 - 15: **return** $\text{TOPK}(\mathbf{r}, K)$
-

3.4.1 The bandwidth bottleneck

The trapdoor evaluation to produce $\mathbf{r}'$ accounts for over 95% of the computation of our client implementation (Appendix D.3). Notably $\mathbf{r}'$ does not depend on the server's answer. The server's computation is a single large matrix-vector product. Beyond these computations, the latency is also depends on the data-transfer time, specifically, the client's download of $\mathbf{A}$ (upload is $< 0.2\text{MB}$, Table 7). As we describe next, both endpoints can perform all their critical computations asynchronously while streaming $\mathbf{A}$.

Client. The client begins computing $\mathbf{r}' = \tilde{\mathbf{q}}\mathbf{Z}^T$ as soon as $\tilde{\mathbf{q}}$ is formed. Our implementation tiles the entire matrix with $d \times d$ square TDMs using the construction from [4]. There are cache-friendly variants that may perform much better (see Section 5.4), however, this is not strictly needed because real bottleneck is elsewhere. Even with our un-optimized TDM implementation (Appendix D.3), the actual *download* of the results takes longer on any realistic network settings (Fig. 8).

One advantage of the this tiling is that neither the TDM computation, nor the download has to completely finish for the client to begin decrypting. The client streams the download in chunks, while simultaneously evaluating the corresponding TDM masks. This enables the client to the client can decrypt chunk of $\mathbf{A}$ as it arrives, and keep a running top K , instead of buffering all N scores. Ultimately the client-side latency is almost entirely governed by the download latency

Server. The server carries the heaviest computation, however, it is solely composed of matrix-

vector products and is highly parallelizable at multiple levels, including both data and pipelines parallelisms [33]. For instance, the matrix can be partitioned freely across rows (documents) into chunks with s blocks each, while parallelizing the computation of each chunk. Since the computation of each block is independent, no synchronization is needed across them. The server returns each block as soon as it is ready, allowing the client’s decryption to continue progressing. This allows for the computation of each block, which is a single matrix-vector products to also benefit from the usual tensor parallelism and other standard optimizations similar to those in neural networks [69].

These points mainly serve to illustrate the bandwidth bottleneck and are by no means exhaustive. For instance, further parallelism is available through architecture-specific features such as SIMD vectorization and cache blocking.

4 Higher-Degree EMVP

As mentioned in the introduction, constructing LEXICURE-2 requires evaluating quadratic functions on encrypted data, whereas the original EMVP protocol supports only linear functions. In this section we describe EMVP², our extension of EMVP to quadratic and higher-degree functions.

Computing a degree- δ function using EMVP is possible by describing it as a linear function of the tensor $\mathbf{q}^{\otimes \delta}$, but this requires applying EMVP to a vector of dimension d^δ . Instead, we present a “white-box extension” of [4] for which communication grows essentially exponentially in $\delta/2$. This lets us compute the quadratic functions we need ($\delta = 2$) with very small overhead. This extension may also be useful beyond the current application to encrypted lexical search; see the discussion in Section 4.2 below. Below we describe the simplest instance of our idea: computing a bilinear form with one scalar output, $(\mathbf{v}, \mathbf{w}) \mapsto \mathbf{w}\mathbf{M}\mathbf{v}^T$. The general construction follows by expressing a degree- δ function of $\mathbf{v}$ as a bilinear form applied to the degree- $(\delta/2)$ tensor, $f(\mathbf{v}) = \mathbf{v}^{\otimes \delta/2} \mathbf{M} (\mathbf{v}^{\otimes \delta/2})^T$.

4.1 EMVP²: Encrypted Bilinear Forms

In this setting, the client wants to store an encrypted matrix $\mathbf{M} \in \mathbb{F}^{d \times d}$ on the server and privately compute the function $f_{\mathbf{M}} : \mathbb{F}^{2d} \rightarrow \mathbb{F}$, where $f_{\mathbf{M}}(\mathbf{v}, \mathbf{w}) = \mathbf{w}\mathbf{M}\mathbf{v}^T \in \mathbb{F}$.

EMVP² closely follows the structure of [4] but uses coding on both the left and right sides of the matrix. That is, the matrix $\mathbf{M}$ is encrypted as $\hat{\mathbf{M}} = \mathbf{D}^T \mathbf{M} \mathbf{D} + \mathbf{Z}$, where $\mathbf{Z}$ is a trapdoored pseudorandom matrix [12, 94]. Query vectors are encrypted exactly as in EMVP. On an encrypted query $(\hat{v}, \hat{w})$, the server breaks $\hat{\mathbf{M}}$ into $s \times s$ blocks and returns $a_{i,j} = \hat{w}_i \hat{\mathbf{M}}_{i,j} \hat{v}_j^T$ for all $i, j \in [s]$. Decryption is a direct analog of EMVP decryption.

EMVP² is described explicitly in Fig. 4. It extends directly to rectangular bilinear forms $\mathbf{M} \in \mathbb{F}^{d_1 \times d_2}$ by using two codes $(\mathbf{C}_1, \mathbf{D}_1)$ and $(\mathbf{C}_2, \mathbf{D}_2)$ with the appropriate dimensions.

Lemma 4.1 (EMVP² Correctness). *The decryption formula $\alpha_w \mathbf{A} \alpha_v^T - \tilde{\mathbf{w}} \mathbf{Z} \tilde{\mathbf{v}}^T$ as described in Fig. 4 correctly outputs the result of the bilinear form $\mathbf{w}\mathbf{M}\mathbf{v}^T$.*

Proof sketch. Consider first the simplified case where all the α scalars are equal to 1. In this case we have $\hat{\mathbf{v}} = \tilde{\mathbf{v}} = (\mathbf{v} | 0^k) + \mathbf{r}_v \mathbf{C}$ and $\hat{\mathbf{w}} = \tilde{\mathbf{w}} = (\mathbf{w} | 0^k) + \mathbf{r}_w \mathbf{C}$. Moreover, the server computes all the

Parameters: $d, k, n = d + k, s, b$ (same as in [4]).

KeyGen. Client chooses an $[n, k]$ -code/dual $(\mathbf{C}, \mathbf{D})$ and a trapdoored pseudorandom matrix $\mathbf{Z}$.

Encrypt-Matrix($\mathbf{M} \in \mathbb{F}^{d \times d}$). Output $\hat{\mathbf{M}} = \mathbf{D}^T \mathbf{M} \mathbf{D} + \mathbf{Z}$.

Encrypt-query($\mathbf{v}, \mathbf{w} \in \mathbb{F}^d$). Each of the vectors $\mathbf{v}$ and $\mathbf{w}$ is encrypted as in [4]:

- The client draws two codewords $\mathbf{c}_v = \mathbf{r}_v \mathbf{C}$, $\mathbf{c}_w = \mathbf{r}_w \mathbf{C}$, and $2s$ random invertible scalars $\alpha_{v,i}, \alpha_{w,i} \in \mathbb{F}^*$ for $i = 1, \dots, s$.
- Setting $\tilde{\mathbf{v}} = (\mathbf{v}|0^k) + \mathbf{c}_v$, the client multiplies each block of $\tilde{\mathbf{v}}$ by $\alpha_{v,i}$ and concatenates the results to form $\hat{\mathbf{v}}$.
- Setting $\tilde{\mathbf{w}} = (\mathbf{w}|0^k) + \mathbf{c}_w$, the client multiplies each block of $\tilde{\mathbf{w}}$ by $\alpha_{w,i}$ and concatenates to form $\hat{\mathbf{w}}$.

The client sends the encrypted query $(\hat{\mathbf{w}}, \hat{\mathbf{v}})$ to the server and retains the vectors $\tilde{\mathbf{v}}, \tilde{\mathbf{w}}$ and $\boldsymbol{\alpha}_v = (\alpha_{v,1}^{-1}, \dots, \alpha_{v,s}^{-1})$, $\boldsymbol{\alpha}_w = (\alpha_{w,1}^{-1}, \dots, \alpha_{w,s}^{-1})$ for decryption.

Encrypted bilinear form. Server breaks $\hat{\mathbf{M}}$ into s^2 blocks $\hat{\mathbf{M}}_{i,j} \in \mathbb{F}^{b \times b}$, computes the s^2 products $a_{i,j} = \hat{w}_i \hat{\mathbf{M}}_{i,j} \hat{v}_j^T$, and sends the matrix $\mathbf{A} = [a_{i,j}]$ to the client.

Decryption. The client outputs $\boldsymbol{\alpha}_w \mathbf{A} \boldsymbol{\alpha}_v^T - \tilde{\mathbf{w}} \mathbf{Z} \tilde{\mathbf{v}}^T$.

Figure 4: The EMVP² protocol for encrypted bilinear forms. The codes $(\mathbf{C}, \mathbf{D})$ are unchanged from [4].

s^2 partial block products so that adding them all yields the final answer. Hence we have

$$\begin{aligned}
\mathbf{1} \cdot \mathbf{A} \cdot \mathbf{1}^T &= \hat{\mathbf{w}} \hat{\mathbf{M}} \hat{\mathbf{v}}^T = \tilde{\mathbf{w}} \hat{\mathbf{M}} \tilde{\mathbf{v}}^T \\
&= ((\mathbf{w}|0^k) + \mathbf{r}_w \mathbf{C})(\mathbf{D}^T \mathbf{M} \mathbf{D} + \mathbf{Z})((\mathbf{v}|0^k) + \mathbf{r}_v \mathbf{C})^T \\
&\stackrel{(i)}{=} (\mathbf{w}|0^k) \mathbf{D}^T \mathbf{M} \mathbf{D} (\mathbf{v}|0^k)^T + \tilde{\mathbf{w}} \mathbf{Z} \tilde{\mathbf{v}}^T \\
&\stackrel{(ii)}{=} \mathbf{w} \mathbf{M} \mathbf{v}^T + \tilde{\mathbf{w}} \mathbf{Z} \tilde{\mathbf{v}}^T.
\end{aligned}$$

Equality (i) holds because all terms containing $\mathbf{C} \mathbf{D}^T$ or $\mathbf{D} \mathbf{C}^T$ vanish, and equality (ii) holds because $\mathbf{D} = [\mathbf{I} | \mathbf{D}']$.

The general case follows by observing that the multiplications $\boldsymbol{\alpha}_w \mathbf{A} \boldsymbol{\alpha}_v^T$ on decryption exactly undo the multiplications by the α 's during query encryption. $\square$

Since encryption is exactly as in the EMVP protocol of [4], security follows directly under the same assumption.

Lemma 4.2. [EMVP² Security] *The protocol in Fig. 4 is a semantically secure bilinear form protocol under the security of the PRF used for derivation from the secret key, the TDM construction, and the 1D-SLSN assumption of [4].* $\square$

4.1.1 Optimizations for quadratic forms

The protocol can be optimized along several axes when used to compute quadratic forms $\mathbf{q} \mathbf{M} \mathbf{q}^T$:

Shorter upload. In this case, $\mathbf{v} = \mathbf{w} = \mathbf{q}$, so there is no need to encrypt the query twice. The client sends a single EMVP-encrypted query $\hat{\mathbf{q}}$, which the server uses as both $\hat{\mathbf{v}}$ and $\hat{\mathbf{w}}$.

Shorter download. Similarly, in this case the client uses $\alpha_v = \alpha_w$ for decryption. This means that the scalars $a_{i,j}$ and $a_{j,i}$ returned by the server are multiplied on decryption by the same scalar $\alpha_{v,i}^{-1} \alpha_{v,j}^{-1}$ before being added together. Instead of sending both scalars, the server can send the single value $a'_{i,j} = a_{i,j} + a_{j,i}$, and the client can compute $\sum_{i \leq j} \alpha_{v,i}^{-1} \cdot \alpha_{v,j}^{-1} \cdot a'_{i,j}$ (instead of explicitly computing $\alpha_w \mathbf{A} \alpha_v^T$).

Applying both optimizations yields communication of only $n + \binom{s+1}{2}$ field elements.

Less server storage. When the quadratic form is itself symmetric, $M[i, j] = M[j, i]$, we can also save on server storage. This optimization requires a *symmetric* trapdoored pseudorandom matrix, which can be obtained generically from any trapdoored pseudorandom matrix, as described below. Once we have a symmetric trapdoored pseudorandom matrix $\mathbf{Z}$, the encrypted matrix $\hat{\mathbf{M}} = \mathbf{D}^T \mathbf{M} \mathbf{D} + \mathbf{Z}$ is also symmetric. Hence, the server stores only its upper triangular part, consisting of $\binom{n+1}{2}$ field elements rather than n^2 .

In that case, the server also computes only about half of the scalars, since $a_{i,j} = a_{j,i}$.

Symmetric trapdoored pseudorandom matrices. To construct a symmetric trapdoored pseudorandom matrix, start from any square trapdoored pseudorandom matrix $\mathbf{Z}$ and set $\mathbf{Z}' = \mathbf{Z} + \mathbf{Z}^T$.

- Clearly, if $\mathbf{Z}$ is a pseudorandom square matrix then $\mathbf{Z}'$ is a pseudorandom symmetric matrix.
- Moreover, by the matrix transposition principle, if an algorithm can use the trapdoor to compute the function $\mathbf{x} \mapsto \mathbf{x} \mathbf{Z}$ quickly, then there exists a similarly fast algorithm that uses the same trapdoor to compute $\mathbf{x} \mapsto \mathbf{x} \mathbf{Z}^T$. We therefore obtain a fast algorithm that, given the trapdoor, computes $\mathbf{x} \mapsto \mathbf{x} \mathbf{Z}' = \mathbf{x} \mathbf{Z} + \mathbf{x} \mathbf{Z}^T$.

There may be symmetric trapdoored-matrix constructions that are faster than this generic construction and perhaps remove its factor-of-two overhead relative to standard trapdoored matrices. Finding such constructions is an interesting open problem.

4.2 Use Cases for Higher-Degree Functions

Our higher-degree EMVP extension significantly enlarges the range of applications to which the “poor man’s FHE” of EMVP applies. Beyond the search use-case that we describe here, other examples may include outsourcing the attention layers of a transformer architecture (which are quadratic forms in the latent state), statistics, multivariate cryptographic functions, etc.

One can also use EMVP to reduce the degree of a complicated function before applying FHE. Outsourcing a degree- δ function f can be implemented by writing it as a composition $f = f_1 \circ f_2$ for f_1 of degree δ_1 and f_2 of degree $\delta_2 = \delta - \delta_1$. One can then outsource f using EMVP $^{\delta_1}$ to compute f_1 , and using HE of degree $\delta_2 + 1$ to compute the EMVP decryption followed by f_2 .

This suggests a library of fast, privately outsourced low-degree functions that can be called as needed and composed into more complex computations. This ability promotes EMVP-based techniques from a narrow, special-purpose primitive to a useful general-purpose tool, on par with other MPC tools such as garbled circuits or FHE.

5 Lexicure-2: Two-Round Protocol with Sublinear Communication

The LEXICURE-1 protocol returns to the client the entire encrypted vector of N similarity scores, resulting in $\Omega(N)$ communication bandwidth. Although our implementation batches and streams this computation, its bandwidth can still be prohibitive for large corpora and weak clients.

Our second variant, LEXICURE-2, reduces this bandwidth by partitioning the documents into G groups of size N/G and using two rounds. (G is a parameter that can be optimized.) In the first round, the client identifies a small set of groups likely to contain the top K matches. In the second round, it privately retrieves the individual scores from only those groups and selects the top K documents among them.

The challenge is how to identify the likely candidate groups in the first step. Consider the vector $\mathbf{r}$ of similarity scores and let $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_G$ be the partial similarity-score vectors corresponding to the G groups. What we want is for the server to compute (on encrypted data) the largest score in each of these sub-vectors (i.e., their l_∞ -norm), so that after decryption the client can choose only those groups containing high enough scores. But the l_∞ norm is a hard function to compute on encrypted data, and it seems that this would require heavy FHE tools that we would like to avoid.

Here we take a different approach. We approximate the l_∞ norm using the (squared) l_2 -norm, which can be evaluated using our EMVP² protocol, and validate experimentally that this approximation is good enough. Namely, we measured the drop in accuracy on our datasets when using squared l_2 -norm instead of l_∞ in this two-step protocol. Note that the smaller G is, the less communication we need in the first step of the protocol, but also the groups are larger so the squared l_2 -norm approximation is worse. Our experiments show that we can find a number $G = O(\sqrt{N})$ that saves a lot on bandwidth while maintaining good accuracy. In the first round below, we use its quadratic-form case to compute group scores; in the second round, we use its rectangular bilinear-form case for private group retrieval.

Why not linear aggregates? Before describing our protocol, we remark that purely linear functions such as averaging seem insufficient for implementing the multi-step approach of LEXICURE-2. For a group of N/G documents with scores $s_1, \dots, s_{N/G}$, the average score $\bar{s} = \frac{1}{N/G} \sum_i s_i$ does not reliably distinguish between a group containing a single document with high score versus multiple documents with scores within the approximation error range. From the CountSketch tail bound (Section 3.2), the approximation error for each score is $O(\|\mathbf{a}\| \|\mathbf{b}\| / \sqrt{d})$, making it difficult to detect a single relevant document when N/G is large.

5.1 Document Partitioning

Fix a group count $G \ll N$, and partition the N documents into G groups of N/G documents each. In the first round, the server computes an aggregate score for each group and sends only G encrypted group scores to the client. The client decrypts these G scores and chooses the top K_G candidate groups. (K_G is another parameter, kept to a small fraction of G ; in our evaluation it never exceeds $K_G/G = 5\%$.) The natural way to read K_G is as that fraction: the client ends up downloading scores for a K_G/G fraction of the corpus, so $K_G = 100$ out of $G = 10^4$ groups means about 1% of the documents. In the second round, the client privately retrieves the similarity scores of the individual documents in all those groups.

The client receives G group scores in the first round and $K_G N/G$ document scores in the

second. Ignoring the cryptographic expansion, this is minimized by setting $G \approx \sqrt{K_G N}$, yielding $O(\sqrt{K_G N})$ downloaded scores.

The same filtering idea could be applied recursively, similar to recursive PIR constructions [23]. This reduces the overall communication, at the price of more rounds and worse accuracy. In practice we find that a single level of partitioning suffices for our target datasets.

5.2 Round One: Group-Score Aggregation

We use EMVP² to evaluate the squared Euclidean norm for every group. Let $\mathbf{M}$ be the matrix with the densified BM25 vectors, and let $\mathbf{M}_i$ be the i 'th block of rows in $\mathbf{M}$ of size (N/G) , representing the documents in the i 'th group. During offline setup, the client forms

$$\mathbf{B}_i = \mathbf{M}_i^T \mathbf{M}_i \in \mathbb{F}^{d \times d} \quad (5.1)$$

for each $i \in [G]$. It encrypts all the $\mathbf{B}_i$ matrices using the same code $(\mathbf{C}, \mathbf{D})$ but independent trapdoored pseudorandom masks $\mathbf{Z}_i$, then uploads the encrypted matrices $\hat{\mathbf{B}}_i$ to the server.

At query time, the client sketches the query as $\mathbf{q} \in \mathbb{F}^d$, encrypts it once with EMVP², and sends $\hat{\mathbf{q}}$ to the server. The server evaluates the same encrypted query on every $\hat{\mathbf{B}}_i$ and returns G encrypted answers. After decryption, the client obtains

$$\gamma_i = \mathbf{q} \mathbf{B}_i \mathbf{q}^T = \|\mathbf{q} \mathbf{M}_i^T\|_2^2 \quad (5.2)$$

for each group and selects the K_G largest values.

Since each $\mathbf{B}_i$ is symmetric and the same query is used on both sides, we apply the quadratic-form optimizations from Section 4.1. The first round uploads n_d field elements for the encrypted query and downloads $G \binom{s+1}{2}$ field elements for the answers. The server work is $\Theta(G \cdot n_d^2)$ field multiplications. Client work is much less, and most of it can be done while waiting for the server response, see details in Section 5.4 below.

5.3 Round Two: Private Group Retrieval

The client must next recover the document scores in the selected groups without revealing their indices. We implement this step using a simple PIR protocol, again using EMVP². The offline setup arranges the document matrix for this purpose. Denote $L = N/G$, and for every within-group position $j \in [L]$, define

$$\mathbf{P}_j = ((\mathbf{M}_1)_{j,*}; \dots; (\mathbf{M}_G)_{j,*}) \in \mathbb{F}^{G \times d}. \quad (5.3)$$

Thus, row i of $\mathbf{P}_j$ is the sketch of the j th document in group i . The client encrypts each rectangular form $\mathbf{P}_j$ with EMVP² during setup and stores the encrypted forms at the server.

In the second round of the query protocol, the client creates a one-hot selector $\mathbf{e}_i \in \{0, 1\}^G$ for every selected group i , encrypts it, and sends it to the server. Using the encrypted query retained from round one, the server evaluates

$$\mathbf{e}_i \mathbf{P}_j \mathbf{q}^T = \mathbf{q} (\mathbf{M}_i)_{j,*}^T \quad \text{for every } j \in [L]. \quad (5.4)$$

These are exactly the L document scores in group i . This is a rectangular bilinear form: one encrypted input selects a group and the other supplies the search query. EMVP² therefore hides both the query and the selected group.

For K_G groups, the client sends K_G encrypted selectors in round two; the encrypted query was already sent in round one. The server can batch the selectors into matrix multiplication operations rather than K_G independent matrix-vector products, improving locality and vectorization [41]. With constant-rate codes, these selectors contain $O(GK_G)$ field elements. Assuming split parameter s for both inputs to the rectangular EMVP² computation, the round-two answers contain $s^2 K_G N/G$ field elements. The client decrypts all $K_G L$ returned scores and selects the top K documents. The server learns nothing beyond the fact that K_G groups were retrieved.

Combining the two rounds, the total online bandwidth and server storage are

$$\begin{aligned} \text{Upload} &= n_d + K_G n_G = \Theta(d + GK_G), \\ \text{Download} &= G \binom{s+1}{2} + s^2 K_G \frac{N}{G}, \\ \text{Server storage} &= G \binom{n_d+1}{2} + \frac{N}{G} n_G n_d \end{aligned} \tag{5.5}$$

field elements, where n_d and n_G are the encoded lengths for vectors of dimensions d and G , respectively. The storage expression uses the symmetric-matrix optimization for the first-round matrices. For constant-rate codes, the second-round storage term is independent of G , whereas the first-round term grows linearly with G . For fixed s , choosing $G = \Theta(\sqrt{K_G N})$ gives download complexity $O(s^2 \sqrt{K_G N})$.

5.4 Batching TDM Evaluation

The two-round protocol requires the client to decrypt G group scores and $K_G N/G$ individual document scores. With a square $n \times n$ trapdoored pseudorandom matrix (TDM), naively decrypting each group score requires evaluating a quadratic mask of the form $\tilde{\mathbf{q}} \mathbf{Z} \tilde{\mathbf{q}}^T$. Using NTT/FFT-based TDM constructions [4] takes $O(n \log n)$ time per matrix, for a total of $O(Gn \log n)$ in round one and $O(K_G(N/G)n \log n)$ in round two. As in LEXICURE-1, this TDM evaluation is expected to dominate the client-side computation, but can be done asynchronously to the download for that round (although LEXICURE-2 is designed to require a smaller download).

One way to improve this computation is to switch to better TDM constructions. For example, Benhamouda et al. describe in [4, Section 5.1.2] other TDM constructions that avoid NTTs and run in time linear in the matrix dimensions, particularly for rectangular (non-square) matrices. Both rounds can benefit from these constructions by batching the data across the appropriate axis. For round one in particular, a rectangular mask can be used across different groups, since all of them share the encrypted query vector.

First-round decryption. Instead of masking each group matrix with an independent square TDM $\mathbf{Z}_i \in \mathbb{F}^{n \times n}$, one may batch B groups and mask them with one rectangular TDM $\mathbf{Z} \in \mathbb{F}^{Bn \times n}$. This is possible because all groups share the same encrypted query vector. Using its trapdoor, the client can compute $\mathbf{Z} \tilde{\mathbf{q}}^T$ in $O(Bn)$ time. The rest of the decryption process is also linear in the result. Therefore, the total client runtime in this round becomes $O(Gn)$, eliminating the $\log n$ factor. Unfortunately, this construction from [4, Section 5.1.2] does not yield symmetric trapdoor matrices, so using it will preclude the optimization from Section 4.1.1 that halves the server storage. Hence, our resource estimates in Section 6.3.1 are based on the square TDM construction [4] that we implemented for LEXICURE-1. Nevertheless, TDM constructions are an active research area

(cf. [12, 94]) and better constructions will likely emerge. This would let us keep the factor-of-two savings in server storage while improving the client processing load.

Second-round decryption. The encrypted forms $\mathbf{P}_j$ are rectangular, so their masks can be evaluated using the same linear-time rectangular TDM construction rather than an NTT for every returned score.

5.5 Setup Complexity

Because this work considers static datasets, we view the index construction where the matrices are encrypted as a one-time offline phase. For the first round, the client forms and encrypts the G matrices $\mathbf{B}_i = \mathbf{M}_i^T \mathbf{M}_i$, which contain a total of Gd^2 plaintext field elements. For the second round, it reorganizes $\mathbf{M}$ into the $L = N/G$ matrices $\mathbf{P}_j$ and encrypts them with EMVP²; together, these matrices contain Nd plaintext field elements. The client can form and encrypt one matrix or batch at a time, so it need not hold the entire index in memory. These independent operations can also be parallelized.

6 Evaluation

In this section, we demonstrate that LEXICURE achieves accuracy comparable to unencrypted BM25 while requiring significantly lower communication and client resources than existing secure lexical search systems.

Our key findings are:

- **Accuracy:** LEXICURE achieves **96%** of plaintext BM25 performance on average on both MRR and Recall (hit-rate), with most datasets exceeding 95%. On the largest dataset, MS MARCO, LEXICURE retains **99.8%** of the plaintext Recall.
- **Communication cost:** LEXICURE clearly outperforms all baselines, requiring **2.6–7.2 $\times$ lower** communication than the second-best alternative for each dataset. The only baseline that achieves comparable accuracy to LEXICURE is Inv-O-BM25, an ORAM-based inverted index. Even with communication-focused optimizations for Inv-O-BM25, such as storing the position map and the term vocabulary on the client, LEXICURE’s uploads are at least **three orders of magnitude smaller**.

LEXICURE already sets a new benchmark for communication cost. Nevertheless, communication remains a primary bottleneck for user-perceived latency. To address this, in Section 6.3.1, we present further improvements attainable by LEXICURE-2 through certain tradeoffs.

- **User-perceived latency:** LEXICURE’s compute is entirely highly parallelizable arithmetic. The server computes the results in chunks, allowing the client to stream and decrypt them asynchronously. Thus, for a well-optimized implementation, the user-perceived latency is largely governed by the client’s network (presumably slower than the server’s). On typical networks such as cable and 5G, LEXICURE improves data transfer latency by up to **20-30x**. Even on high-speed broadband connections, LEXICURE wins by up to **10x** (Fig. 8). In sharp contrast, ORAM’s large uploads make it unusable on asymmetrical networks. As seen in Fig. 8, Inv-O-BM25 is sometimes worse than the client simply downloading the entire index to perform the search locally.

- **Client storage:** LEXICURE requires **less than 1 KB** of persistent client storage, compared to hundreds of MB for ORAM-based approaches and several MB for HE-Cluster. Notably, LEXICURE does not require the use of an embedding model, eliminating the question of securely training and storing such a model entirely.
- **Server storage:** LEXICURE’s server storage (227GiB–1.3TiB) is comparable to HE-Cluster (FHE + Tiptoe [45]), but significantly higher than Inv-ORAM. This is how we absorb the inevitable cost of privacy while allowing for the above gains on all client-facing metrics. We target the asymmetric setting of the server having much more capacity and powerful hardware than the client. Similar to Tiptoe’s design, our server provides both compute and storage. In contrast, most ORAM-based servers provide only storage [81].

In the natural setting of a large-scale server and a typical personal device as the client, LEXICURE is the clear winner.

These experiments are organized as below.

Section 6.2: Comparisons of LEXICURE against existing encrypted search baselines comprising other natural approaches for secure lexical search.

Section 6.3: Standalone evaluation of LEXICURE on 16 diverse datasets and of LEXICURE-2 (Section 6.3.1) on large datasets.

6.1 Experimental Setup

For all the experiments we used `pyserini` [61] to generate the BM25 sparse vector embeddings with default parameters. We tokenized LEXICURE and all inverted-index-derived baselines identically, using `pyserini`’s `DefaultEnglishAnalyzer`. For every dataset, we uniformly fix the densifier dimension $d = 10^4$ and, for LEXICURE-2 experiments, the number of groups $G = 10^4$. Our EMVP and EMVP² instantiations guarantee 128-bit security with the detailed settings provided in Appendix D.1.

6.1.1 Datasets

We used 16 datasets consisting of the 15/18 publicly available ones from the BEIR benchmark [91] and the TripClick dataset [80]. Table 5 summarizes the attributes of each dataset. For datasets with multiple query splits, we used the `test` split for evaluations. The vocabulary sizes (unique terms) range from approximately 20K to over 3M terms, highlighting the necessity of our densifier approach.

6.1.2 Evaluation Metrics

We assess search quality on three standard information retrieval metrics at a top 10 cutoff using the `trec_eval` tool [73]. These consist of *MRR*, which measures the rank of the first relevant document, *Recall*, which measures the hit-rate, and *NDCG*, which grades the overall ranking. Formal definitions of these metrics are provided in Appendix B.

In Section 6.2, we stick to architecture-independent metrics that can reliably be reproduced on any system. Specifically, we assess each approach on accuracy, communication per query (upload/download sizes), client storage (persistent), and server storage. Metrics such as latency and RAM usage are highly influenced by architecture-dependent optimizations. LEXICURE in partic-

Table 5: Dataset statistics. Unique terms are extracted from Pyserini Lucene indices using the default text analyzer.

Dataset	Domain	Queries	Docs	Terms
ArguAna [97]	Misc.	1,406	8,674	23,895
Climate-FEVER [34]	Wikipedia	1,535	5,416,593	3,293,621
CQADupStack [47]	StackEx.	13,145	457,199	1,098,180
DBPedia [44]	Wikipedia	400	4,635,922	3,351,459
FEVER [92]	Wikipedia	6,666	5,416,568	3,293,639
FiQA [63]	Finance	648	57,600	66,977
HotpotQA [100]	Wikipedia	7,405	5,233,329	2,644,892
MSMARCO [71]	Misc.	6,980	8,841,823	2,660,824
NFCorpus [10]	Bio-Med.	323	3,633	22,111
NQ [59]	Wikipedia	3,452	2,681,468	997,027
Quora [84]	Quora	10,000	522,931	69,597
SCIDOCS [25]	Scientific	1,000	25,657	63,604
SciFact [98]	Scientific	300	5,183	28,865
Touche-2020 [7]	Misc.	49	382,545	525,540
TREC-COVID [96]	Bio-Med.	50	171,331	202,648
TripClick [80]	Health	1,143	1,523,878	897,936

ular benefits heavily from such optimizations due to both the client and the server consisting of embarrassingly parallelizable operations.

6.2 Comparison to existing approaches

We compared LEXICURE against the following baselines on all seven datasets in our evaluation with more than one million documents. Note that all three approaches use static indexes. Supporting updates introduces additional overhead, as discussed in Section 7.

Inv-ORAM. Implements encrypted lexical search using a TF-IDF inverted index over ORAM for privacy [101, 81]. Inv-ORAM treats the server as untrusted storage and performs the search computation on the client. To avoid leaking query length and keyword frequency, queries and responses are padded to a fixed maximum size L . A higher L increases communication and client computation, while a lower L degrades search quality by truncating longer posting lists. The original implementation of [101] keeps L fixed to 10,000. We found that this caused the quality to rapidly deteriorate with corpus size. We instead set L to 1% of the corpus size for all datasets, resulting in accuracy comparable to the plaintext TF-IDF. We retained the original settings for all other parameters, including the ORAM and query-length padding.

Inv-O-BM25. To provide a fairer comparison to LEXICURE, we modified Inv-ORAM to use BM25 scoring instead of TF-IDF. This does not affect the server/client sizes or the communication size as a function of L but considerably improves accuracy. As before, we kept L fixed to 1% of corpus size, which consistently keeps the accuracy in the same range as LEXICURE and plaintext. We point out that this value of L is not excessive, as it still stays below plaintext. We found that further lowering L , for instance to 0.8% of corpus, resulted in both worse accuracy and communication cost than LEXICURE.

HE-Cluster. Implemented in Compass [101] to perform encrypted semantic search using FHE by extending the clustering-based Tiptoe [45], which was intended for public databases. HE-Cluster uses semantic embeddings which are of much lower dimension than our dense embeddings (768 vs 10^4), but it can, in theory, be extended to use our embeddings. This could potentially improve its accuracy on lexical search but hurts storage and communication. Unfortunately, clustering millions of 10^4 -dim vectors was beyond the capabilities of our machine. Therefore, we evaluated the original semantic version as a stand-in, since, to our knowledge, this is the only FHE-based encrypted-search system with a private index.

6.2.1 Results

In summary, HE-Cluster and Inv-ORAM both perform significantly worse than Inv-O-BM25 and LEXICURE on accuracy and communication. LEXICURE outperforms all the baselines on communication and client memory, while both ORAM baselines perform better on server memory.

Search Quality. On all datasets, LEXICURE and Inv-O-BM25 both perform close to plaintext BM25. LEXICURE slightly outperforms Inv-O-BM25 at all three metrics on HotpotQA and Fever, while, Inv-O-BM25 slightly outperforms LEXICURE on TripClick and DBPedia. The remaining two baselines, fall noticeably below LEXICURE and Inv-O-BM25 on all datasets and metrics.

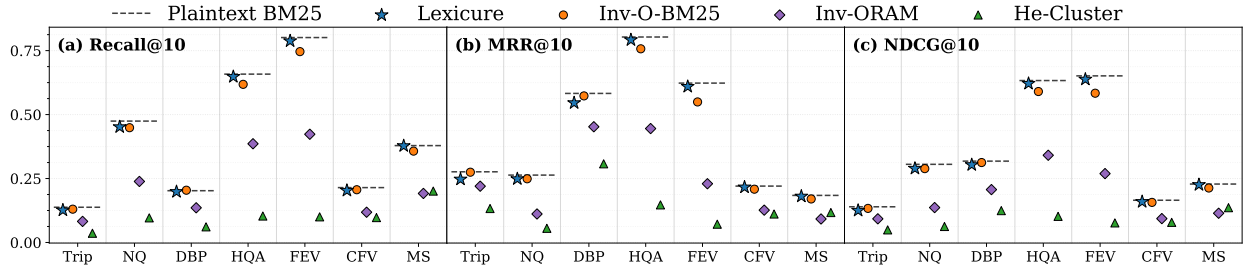


Figure 6: Comparison of Recall@10, MRR@10, and NDCG@10 across methods and datasets. Listed in increasing order of corpus size are TripClick, NQ, DBPedia, HotpotQA, FEVER, Climate-FEVER and MS MARCO.

Communication. All the baselines require a single roundtrip communication round. LEXICURE requires the least communication on every dataset. Inv-O-BM25, the only baseline with comparable accuracy, ranges from $2.6\times$ (DBPedia) to $9.9\times$ (Climate-FEVER) more than LEXICURE. Inv-ORAM’s communication also scales with the query-length, since each query term results in ORAM look-ups. Climate-FEVER contains claim-style queries, in contrast to the short Bing search-log queries in MS MARCO. This is why Inv-ORAM’s communication costs the most on Climate-FEVER. Despite using smaller embeddings than LEXICURE, HE-Cluster costs $4.5\times$ - $10.6\times$ more communication, demonstrating the superiority of using EMVP over FHE.

Effects on user-perceived latency. Communication is typically the bottleneck in settings with a weak client and especially for LEXICURE (Section 3.4.1). The client-side processing in all these approaches scales with the communication cost, making it a justifiable indicator of this latency. To this end, we estimate the data-transfer time for these approaches in Figure 8 on four representative

Table 7: Comparison of per-query communication (in MiB) split into upload (client to server) and download (server to client). Index-DL refers to downloading an AES-256-GCM encryption of the plaintext index.

Dataset ($\downarrow$)	He-Cluster			Inv-O-BM25			Lexicure			Index-DL		
	Up	Down	Total	Up	Down	Total	Up	Down	Total	Up	Down	Total
MSMARCO	1905.4	361.6	2267.0	1038.7	958.8	1997.5	0.15	505.9	506.1	0	4180.7	4180.7
Climate-FEVER	1921.9	309.6	2231.5	1595.6	1472.9	3068.5	0.15	309.9	310.1	0	4532.2	4532.2
FEVER	1937.9	274.6	2212.5	855.1	789.3	1644.4	0.15	309.9	310.1	0	4532.1	4532.1
HotpotQA	1865.9	227.1	2093.0	1134.2	1046.9	2181.1	0.15	299.4	299.6	0	2385.2	2385.2
DBPedia	1648.9	213.1	1861.9	364.2	336.2	700.4	0.15	265.3	265.4	0	5088.0	5088.0
NQ	1027.7	184.0	1211.8	284.1	262.3	546.4	0.15	153.4	153.6	0	1926.5	1926.5
TripClick	765.2	164.0	929.2	174.9	161.4	336.3	0.15	87.2	87.4	0	3309.0	3309.0

networks: 4G, 5G, Cable, and Fiber. The first three have slower upload speeds than download, while Fiber, the fastest, allows 1Gbps symmetric upload and download speeds. LEXICURE’s small upload greatly pays off on the asymmetric links, making it $8.1\text{--}31.5\times$ (cable) and $8.1\text{--}21.6\times$ (5G) faster than the best baselines. Meanwhile, downloading the index (Index-DL) occasionally beats the ORAM baselines despite transferring far more data. As expected, LEXICURE also performs the best on Fiber due to having the smallest total bandwidth.

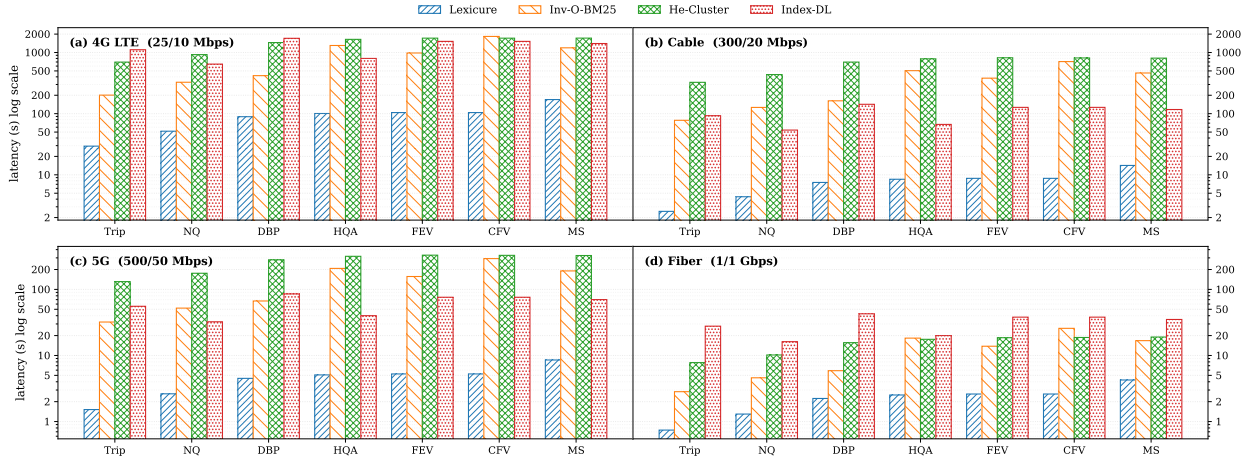


Figure 8: Estimated single-query communication latency by client network type, at download/upload rates of 25/10 Mbps (4G LTE), 300/20 Mbps (Cable), 500/50 Mbps (5G), and 1/1 Gbps (Fiber). Index-DL represents the cost of downloading the entire lexical index.

Memory. LEXICURE needs less than 1 KB of persistent client storage—only the cryptographic keys and a seed for the densifier. This is a substantial improvement over both the baselines. The client-side storage for both Inv-ORAM variants, consisting primarily of the ORAM position map, ranges from 465.8 MiB (NQ) to 1055.0 MiB (MS MARCO) as shown in Table 9. This ORAM map needs to either be stored on the client in persistent memory, or be downloaded for each new session, for instance via a recursive ORAM. Although a recursive ORAM can reduce this client-side persistent memory, it increases the overall communication cost, which is already the main weakness of ORAM.

The cost of this lightweight client in LEXICURE is higher server storage (1317.58 GiB for MS

Table 9: Comparison of persistent client storage. All values in MiB.

Dataset ($\downarrow$)	He-Cluster	Inv-O-BM25	Lexicure
MSMARCO	11.1	1055.0	<1 KB
Climate-FEVER	11.3	938.1	<1 KB
FEVER	11.3	938.1	<1 KB
HotpotQA	10.9	549.8	<1 KB
DBPedia	9.7	545.4	<1 KB
NQ	6.0	465.8	<1 KB
TripClick	4.5	689.0	<1 KB

MARCO), comparable to HE-Cluster (1345.3 GiB) but much larger than Inv-ORAM (12.0 GiB). Inv-ORAM treats the server as an untrusted memory and performs all the computations on the client, whereas both LEXICURE and HE-Cluster use the server to compute on encrypted data.

Table 10: Comparison of server size. All values in GiB.

Dataset ($\downarrow$)	He-Cluster	Inv-O-BM25	Lexicure
MSMARCO	1345.3	12.0	1317.5
Climate-FEVER	1161.8	12.0	807.1
FEVER	1039.0	12.0	807.1
HotpotQA	827.3	12.0	779.8
DBPedia	686.0	6.0	690.8
NQ	369.3	6.0	399.6
TripClick	245.1	12.0	227.1

6.3 Additional evaluations of Lexicure

Detailed discussion of our LEXICURE experiments can be found in Appendix D.3. Below is a short summary with our main findings.

Accuracy across diverse domains. Table 11 presents the accuracy results of LEXICURE against plaintext BM25 on all 16 datasets. LEXICURE achieves overall accuracy of more than 95% of plaintext BM25 (both average and median across all datasets) for all three quality measures (MRR@10, Recall@10, and NDCG@10).

Storage and compute. Our single-threaded C++ client implementation takes under 0.8 seconds for TripClick (1.5M documents) and 4.4 seconds for MS MARCO (8.8M documents), with decryption dominating the latency and scaling linearly with corpus size (Appendix Fig. 13). About 95% of this latency is computing the TDM evaluation (Fig. 14), due to the NTT (FFT over fields) operations in the trapdoor evaluation. This bottleneck can be mitigated through parallelization or cache-friendly batching discussed in Section 5.4. Client persistent storage is minimal with only a few bytes for cryptographic keys and densifier seeds. Server storage ranges from 0.5GB to 1.3TB, depending on corpus size. The server performs s matrix-vector products requiring approximately

Table 11: Evaluation Results: BM25 vs Lexicure

Method ($\rightarrow$)	Recall@10			MRR@10			NDCG@10		
Dataset ($\downarrow$)	BM25	Lexicure	%	BM25	Lexicure	%	BM25	Lexicure	%
ArguAna	0.642	0.631	98.2%	0.193	0.188	97.5%	0.300	0.293	97.8%
Climate-FEVER	0.214	0.203	94.7%	0.220	0.216	98.1%	0.165	0.159	96.5%
CQADupStack	0.354	0.340	96.0%	0.281	0.267	95.1%	0.283	0.270	95.3%
DBPedia	0.202	0.198	98.0%	0.583	0.546	93.7%	0.318	0.304	95.5%
FEVER	0.801	0.788	98.3%	0.623	0.609	97.8%	0.651	0.638	97.9%
FiQA	0.295	0.286	96.8%	0.296	0.280	94.6%	0.236	0.225	95.2%
HotpotQA	0.658	0.648	98.5%	0.803	0.792	98.6%	0.633	0.621	98.2%
MSMARCO	0.379	0.378	99.8%	0.184	0.180	98.1%	0.228	0.226	98.8%
NFCorpus	0.154	0.152	98.4%	0.524	0.525	100.2%	0.322	0.317	98.4%
NQ	0.474	0.452	95.2%	0.263	0.249	94.4%	0.305	0.289	94.7%
Quora	0.889	0.888	99.8%	0.779	0.777	99.7%	0.789	0.786	99.7%
SCIDOCS	0.157	0.150	96.0%	0.261	0.253	97.0%	0.149	0.144	96.7%
SciFact	0.807	0.784	97.1%	0.646	0.632	97.9%	0.680	0.664	97.7%
Touche-2020	0.255	0.214	83.9%	0.744	0.643	86.5%	0.442	0.356	80.6%
TREC-COVID	0.016	0.015	94.9%	0.853	0.831	97.4%	0.595	0.544	91.5%
TripClick	0.138	0.126	91.7%	0.276	0.247	89.4%	0.140	0.125	89.7%
Avg. performance	96.1%			96.0%			95.3%		

$2N \cdot d \cdot f = 80,000 \cdot N$ modular operations. For MS MARCO, this translates to 707 billion modular operations per query, equivalent to 1.4–3.5 TFLOP computation using efficient Solinas prime modular reduction [49]. On modern GPUs such as NVIDIA A100 or H100, this can be expected to take under 20ms per query [75, 74].

6.3.1 Lexicure-2: Communication improvements

Although LEXICURE-1 already uses less communication than the baselines, LEXICURE-2 enables more aggressive bandwidth reduction tradeoffs. Two relevant parameters for LEXICURE-2 are the total number of groups G and the number of retrieved groups K_G . We form the G groups by splitting each corpus into contiguous blocks of the ordering it is distributed with. In our experiments we fixed $G = 10^4$ and varied K_G to get a good accuracy/bandwidth tradeoff. K_G can be viewed as a multiple of K , and lower values of K_G reduce the bandwidth in Round 2 of the protocol but risk the top K documents falling outside the top- K_G groups.

Figure 12 demonstrates this tradeoff on the seven with $> 1\text{M}$ documents, by varying K_G while keeping the other parameters fixed. In particular, $K = 10$, $d = 10^4$ and the underlying EMVP² protocol uses $n = 40,000$, $s = 15$. In Appendix D.5, we present additional tradeoffs between server size and communication cost by varying the EMVP² parameters, which does not affect accuracy. All evaluated settings provide 128-bit security.

Bandwidth vs accuracy results. For MS MARCO (8.8M documents), LEXICURE-2 with $K_G = K = 10$ achieves 100.5% and 103.4% of the plaintext Recall and MRR respectively, requiring only 18.58 MiB of total communication. This is a $27.2\times$ reduction compared to LEXICURE-1’s 506.09 MiB. Raising K_G to $10 \cdot K$ costs about 100 MiB for essentially the same quality ($\geq 100\%$ of plaintext MRR and Recall). For NQ (2.7M documents), $K_G = 2 \cdot K$ achieves about 83% of plaintext MRR and Recall, with 17.13 MiB in communication, a $9.0\times$ reduction to LEXICURE-1.

In Fig. 12, the starred point denotes the lowest-bandwidth configuration that clears at least 80% of plaintext BM25 on *all* the three quality metrics. This threshold still retains much of the effectiveness that is specific to BM25. For reference, plaintext TF-IDF’s Recall@10 alone varies

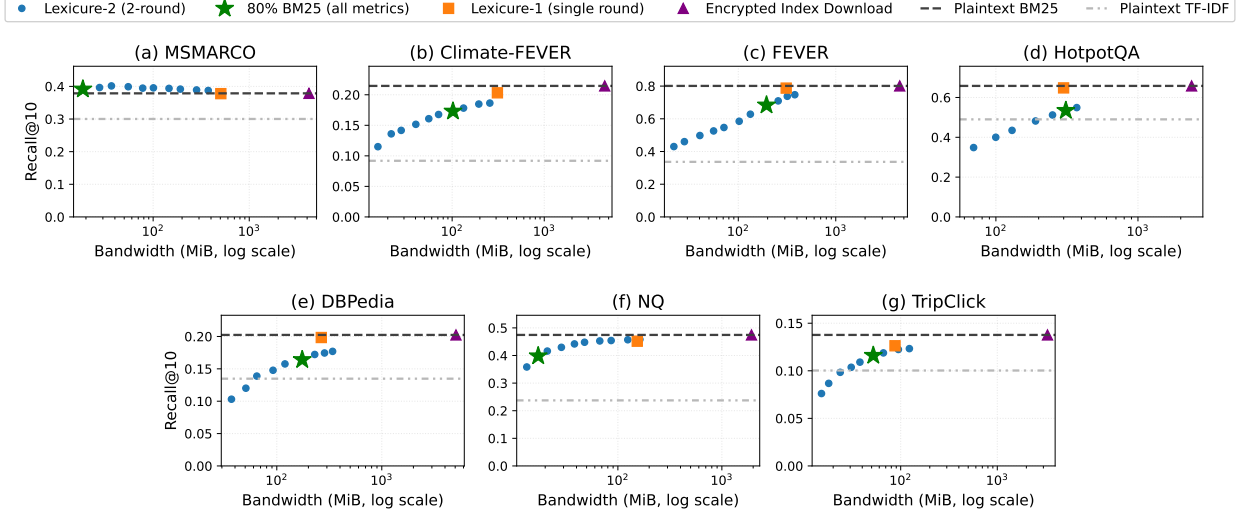


Figure 12: Recall@10 vs. bandwidth tradeoff for LEXICURE-2 on large-scale datasets with $> 1\text{M}$ documents (top and left is better). Datasets are ordered by corpus size (largest to smallest). Each curve shows results for varying K_G (number of groups retrieved), with lower K_G achieving lower bandwidth. The green star marks the lowest-bandwidth configuration achieving $\geq 80\%$ of plaintext BM25 on all three metrics (MRR@10, NDCG@10, Recall@10). The purple triangle indicates a client-side search, which entails downloading an encryption (AES-256-GCM) of the entire plaintext index.

between 42–79% of BM25 on these datasets, dropping further for a secure version (Fig. 6). The K_G required to reach this threshold, and consequently the bandwidth reduction, varies across datasets. MSMARCO and NQ achieve this with $K_G \leq 2 \cdot K$, Climate-FEVER and TripClick require $K_G \geq 15 \cdot K$, and DBPedia, FEVER, and HotpotQA require $K_G \geq 30 \cdot K$. Across six of the seven corpora, this amounts to a $1.5\text{--}27\times$ reduction in communication relative to LEXICURE-1, saving 35–488 MiB per query. The exception is HotpotQA, where this threshold is too high to provide a meaningful reduction. For HotpotQA, a $1.2\times$ bandwidth reduction instead obtains 84.8% and 77.8% of plaintext MRR@10 and Recall@10 respectively.

One important factor that influences this tradeoff is how the G groups are formed. We retained the original document orderings of these datasets and grouped them consecutively, however, more sophisticated approaches may be beneficial here. See Appendix D.5.4 for more discussion.

LEXICURE-2 uses significantly more storage than LEXICURE-1, since it stores the encrypted rectangular forms $\hat{\mathbf{P}}_j$ as well as all the encrypted $\hat{\mathbf{B}}_i$ ’s. Its purpose is to lower communication, and the EMVP parameters set what storage that costs, without affecting accuracy. At the balanced configuration of Fig. 18 ($f_{\text{ndim}} = 2$, $f_{\text{ngroups}} = 4$), the LEXICURE-2 server storage for MS MARCO is 17.13 TiB, with 173.85 MiB of communication at $K_G = 100$ and requiring approximately 8 trillion modular operations. If server storage is less of a constraint, $f_{\text{ndim}} = f_{\text{ngroups}} = 4$ lowers the communication further to 100.68 MiB at 63.36 TiB.

7 Conclusion

This paper presents LEXICURE, a leakage-free encrypted lexical search protocol that achieves high accuracy and state-of-the-art communication cost while maintaining a lightweight client and a

feasible server. Our evaluation on 16 datasets shows that LEXICURE-1 achieves accuracy competitive with plaintext BM25 (typically $> 95\%$) while requiring significantly less bandwidth and client memory than prior encrypted search systems. Using our EMVP² primitive we demonstrate that leakage-free encrypted lexical search leakage-free is possible even in realistic settings of weak client devices.

We view LEXICURE as the initial step toward practical leakage-free encrypted search, without any extended trust assumptions, which has largely been under-explored until this work due to its inherent challenges. The immediate future direction is combining lexical and semantic search, as both can potentially benefit from EMVP². Other extensions include protection against malicious servers using homomorphic MACs [3, 29] and supporting deletions. Additionally, we believe pre-processing techniques such as document clustering [45] could improve the accuracy-bandwidth tradeoffs LEXICURE-2. Some future directions are discussed next.

7.1 Combining Lexical and Semantic Search

Lexical search is rarely used in isolation in modern information retrieval systems. Most production systems combine lexical search (e.g., BM25) with semantic vector search using learned embeddings. Since both types of search can build on EMVP—lexical search through our techniques and semantic search through direct EMVP application—there is substantial room for synergies in combining both types in a single system. A unified system could share encryption infrastructure, amortize communication costs, and potentially use lexical search results to guide semantic search or vice versa.

7.2 Beyond Semi-Honest Security

While semi-honest security is the focus of this work, our protocol can be extended to provide protection against servers that deviate from the prescribed protocol. Specifically, the server’s responses can be authenticated using homomorphic information-theoretic MACs similar to those used in SPDZ [3, 29]. This extension is straightforward because the EMVP construction [4] underlying our protocol naturally supports such MACs. The main overhead would be additional communication for the MAC tags and modest computational cost for verification.

7.3 Dynamic Datasets

As presented, LEXICURE assumes static datasets, treating index construction as a one-time offline operation. Extending LEXICURE to support dynamic updates—adding, modifying, or deleting documents—is important for practical deployments.

Additions. Adding documents to LEXICURE-1 is not too hard, assuming that the server is allowed to know the current size of the corpus and that adding these new documents does not materially change the BM25 ranking of existing ones: The BM25 vector of the new document is computed, then sketched to reduce the dimension, then encoded with the EMVP data code. The only challenge is adding a new row to the trapdoor matrix in order to mask the encoded row. This requires dynamic trapdoored matrix, which is outside the scope of our current work but is not very difficult using the techniques of Benhamouda et al. from [4].¹ (This would require the client to know how many

¹The simplest solution is just to construct a larger matrix to begin with, and use more rows of it as we add more documents.

documents are currently in the corpus, but that information can be retrieved from the server.)

Adding documents in LEXICURE-2 is harder, since we need to also update one of the matrices $B_i = M_f^T M_i$, but this only effect a single group so it could be feasible to fetch the entire group and re-compute its B_i matrix. (Alternatively the added documents could be placed in a stash that is fetch every time, and we could only compute the B_i for these document once the stash reaches a certain size.)

Deletions and updates. Deleting or updating documents is harder, even in LEXICURE-1, since re-using the corresponding row of the trapdoor matrix will leak information. If the server is allowed to learn that a document is deleted, then it can just puncture the matrix, returning for this row the special symbol $\perp$ instead of the usual scalars. Otherwise it has to be the client that keeps the information about deleted rows in an up-to-date cache. Deletion in LEXICURE-2 require updating the relevant B_i matrices. This can be done similar to addition if the server is allowed to learn which group is updated, otherwise it seems that we need to start employing updatable-ORAM for that purpose. Once deletions are possible, updates can be implemented as a deletion follow by addition.

Amortized reconstruction. For datasets with moderate update rates, the client could keep a set of changes up to a point, and periodically reconstruct the entire encrypted index offline, leveraging the one-time setup cost. This can be used for all modifications, and can even handle changes in the BM25 ranking due to changes in the corpus.

7.4 Hybrid EMVP-FHE Approaches

While we deliberately avoided heavy FHE tools to demonstrate how far EMVP techniques alone can reach, a hybrid approach combining EMVP with targeted FHE operations could offer better tradeoffs. For example, homomorphic compression techniques [22, 62, 36, 21] could be applied after EMVP-based score computation to further compress communication. The key challenge is identifying operations where FHE overhead is justified by communication savings, and where EMVP’s efficiency advantages can be preserved.

7.5 Preprocessing and Clustering

The dataset-dependent performance of LEXICURE-2 suggests opportunities for improvement through preprocessing. Our preliminary locality analysis (Appendix D.5.4) indicates that document ordering significantly affects performance: when relevant documents cluster together, the two-round protocol achieves better accuracy because fewer groups need to be retrieved.

Several preprocessing strategies could exploit this observation:

- **Semantic clustering:** Group documents by topic or semantic similarity using standard clustering algorithms (e.g., k -means on document embeddings).
- **Keyword-based clustering:** Group documents sharing common important keywords, similar to the TipToe approach [45].
- **Hybrid clustering:** Combine multiple signals (semantic similarity, keyword overlap, meta-data) to optimize for both lexical and semantic search.

Systematic exploration of preprocessing strategies, including their impact on index construction cost and search accuracy, is an important direction for future work.

7.6 Parameter Optimization

Our evaluation used fixed parameter settings (e.g., $d = 10,000$ for CountSketch dimension, $G = 10,000$ groups for LEXICURE-2). Systematic exploration of the parameter space could improve performance:

- **Adaptive parameters:** Select CountSketch dimension d and number of groups G based on dataset characteristics (vocabulary size, document count, query distribution).
- **Per-query optimization:** Adjust the number of groups K retrieved in the second round based on the distribution of group scores.
- **Batch size tuning:** Optimize TDM batch sizes for specific hardware characteristics (cache sizes, memory bandwidth).

7.7 Document Retrieval

Like most literature on encrypted search, we focus on returning the IDs of selected documents rather than the documents themselves. In practice, clients need to retrieve the actual document content, which can be accomplished using existing PIR [23] or ORAM techniques. However, there is room for synergy and optimization in combining the ID fetching with the document retrieval step:

- The same EMVP infrastructure used for search could potentially be leveraged for document retrieval.
- Batching document retrievals across multiple queries could amortize costs.
- For small documents, embedding document content directly in the encrypted index might be more efficient than separate retrieval.

8 Acknowledgements

The authors are grateful to Yuval Ishai for insightful discussions, for feedback regarding the EMVP² primitive, and for suggesting the symmetric matrix optimization used in Section 4.1.1. The authors also appreciate Mukul Karnik’s encouragement to study encrypted, lexical search.

References

- [1] Adi Akavia, Craig Gentry, Shai Halevi, and Max Leibovich. Setup-free secure search on encrypted data: Faster and post-processing free. *Proc. Priv. Enhancing Technol.*, 2019(3):87–107, 2019.
- [2] Amazon Web Services. Data preparation at scale using Amazon EMR serverless applications or Amazon EMR clusters in studio. Amazon Web Services Documentation, <https://docs.aws.amazon.com/sagemaker/latest/dg/studio-emr-data-preparation.html>. Accessed August 19, 2026.

- [3] Rikke Bendlin, Ivan Damgard, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.
- [4] Fabrice Benhamouda, Caicai Chen, Shai Halevi, Yuval Ishai, Hugo Krawczyk, Tamer Mour, Tal Rabin, and Alon Rosen. Encrypted matrix-vector products from secret dual codes. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS '25*, pages 394–408, New York, NY, USA, 2025. Association for Computing Machinery. Long version available from <https://eprint.iacr.org/2025/858>.
- [5] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15*, page 837–849, New York, NY, USA, 2015. Association for Computing Machinery.
- [6] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *Proceedings 2020 Network and Distributed System Security Symposium*. Internet Society, 2020.
- [7] Alexander Bondarenko, Maik Fröbe, Meriem Beloucif, Lukas Gienapp, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. Overview of Touché 2020: Argument Retrieval. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névél, editors, *Working Notes Papers of the CLEF 2020 Evaluation Labs*, volume 2696 of *CEUR Workshop Proceedings*, September 2020.
- [8] Raphael Bost. $\sigma\phi\sigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154, 2016.
- [9] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1465–1482, 2017.
- [10] Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. A full-text learning to rank dataset for medical information retrieval. In *Proceedings of the 38th European Conference on Information Retrieval (ECIR 2016)*, pages 716–722, 2016.
- [11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, page 603–633, Berlin, Heidelberg, 2022. Springer-Verlag.
- [12] Mark Braverman and Stephen Newman. Practical secure delegated linear algebra with trapdoored matrices. In *Theory of Cryptography Conference*, pages 97–118. Springer, 2025.
- [13] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution. In *27th*

- USENIX Security Symposium (USENIX Security 18)*, page 991–1008, Baltimore, MD, August 2018. USENIX Association.
- [14] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *2011 Proceedings IEEE INFOCOM*, pages 829–837, 2011.
 - [15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679, 2015.
 - [16] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Annual cryptology conference*, pages 353–373. Springer, 2013.
 - [17] T-H Hubert Chan, Kai-Min Chung, and Elaine Shi. On the depth of oblivious parallel ram. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 567–597. Springer, 2017.
 - [18] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.
 - [19] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 577–594, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
 - [20] Tao Chen, Mingyang Zhang, Jing Lu, Michael Bendersky, and Marc Najork. Out-of-domain semantics to the rescue! zero-shot hybrid retrieval models. In *Advances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10–14, 2022, Proceedings, Part I*, page 95–110, Berlin, Heidelberg, 2022. Springer-Verlag.
 - [21] Jung Hee Cheon, Keewoo Lee, Jai Hyun Park, and Yongdong Yeo. Simd-aware homomorphic compression and application to private database query, 2024.
 - [22] Seung Geol Choi, Dana Dachman-Soled, S. Dov Gordon, Linsheng Liu, and Arkady Yerukhimovich. Compressed oblivious encoding for homomorphically encrypted search. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2277–2291. ACM, 2021.
 - [23] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.
 - [24] Sujoy Chowdhury and Karan Singh. Introducing OCI Data Flow pools. Oracle Cloud Infrastructure Blog, <https://blogs.oracle.com/cloud-infrastructure/introducing-oci-data-flow-pools>, September 2023. Accessed August 19, 2026.

- [25] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel Weld. SPECTER: Document-level representation learning using citation-informed transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2270–2282, Online, July 2020. Association for Computational Linguistics.
- [26] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.
- [27] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88, 2006.
- [28] Majid Daliri, Juliana Freire, Christopher Musco, Aécio S. R. Santos, and Haoxiang Zhang. Sampling methods for inner product sketching. *Proc. VLDB Endow.*, 17(9):2185–2197, 2024. Available also from <https://arxiv.org/abs/2309.16157>.
- [29] Ivan Damgard, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
- [30] Marc Damie, Florian Hahn, and Andreas Peter. A highly accurate {Query-Recovery} attack against searchable encryption using {Non-Indexed} documents. In *30th USENIX security symposium (USENIX Security 21)*, pages 143–160, 2021.
- [31] Sankha Das, Rohan Ravi, Nishanth Chandran, and Divya Gupta. Bison: Billion-scale oblivious nearest-neighbor search in milliseconds. In *2026 IEEE 11th European Symposium on Security and Privacy (EuroS&P)*, pages 160–172. IEEE, 2026. Long version available form <https://ia.cr/2026/1343>.
- [32] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. {DORY}: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119, 2020.
- [33] David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. *Commun. ACM*, 35(6):85–98, June 1992.
- [34] Thomas Diggelmann, Jordan Boyd-Graber, Jannis Bulian, Massimiliano Ciaramita, and Markus Leippold. Climate-fever: A dataset for verification of real-world climate claims, 2020.
- [35] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science, ITCS '14*, page 169–182, New York, NY, USA, 2014. Association for Computing Machinery.
- [36] Nils Fleischhacker, Kasper Green Larsen, and Mark Simkin. How to compress encrypted data. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023*

- *42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 551–577. Springer, 2023.
- [37] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.
 - [38] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Paper 2003/216, 2003.
 - [39] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
 - [40] Google Cloud. BigQuery documentation. Google Cloud Documentation, <https://docs.cloud.google.com/bigquery/docs>. Accessed August 19, 2026.
 - [41] Kazushige Goto and Robert A van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):1–25, 2008.
 - [42] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: private inference on transformers. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22*, Red Hook, NY, USA, 2022. Curran Associates Inc.
 - [43] David Harvey and Joris van der Hoeven. Faster integer multiplication using plain vanilla fft primes. *Mathematics of Computation*, 88(315):501–514, 2019.
 - [44] Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. Dbpedia-entity v2: A test collection for entity search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '17*, pages 1265–1268. ACM, 2017.
 - [45] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th symposium on operating systems principles*, pages 396–416, 2023.
 - [46] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and Attila A. Yavuz. Hardware-supported oram in effect: Practical oblivious search and update on very large dataset. *Proceedings on Privacy Enhancing Technologies*, 2019:172 – 191, 2018.
 - [47] Doris Hoogeveen, Karin M Verspoor, and Timothy Baldwin. Cqadupstack: A benchmark data set for community question-answering research. In *Proceedings of the 20th Australasian document computing symposium*, pages 1–8, 2015.
 - [48] Alexander Hoover, Ruth Ng, Daren Khu, Yao’An Li, Joelle Lim, Derrick Ng, Jed Lim, and Yiyang Song. {Leakage-Abuse} attacks against structured encryption for {SQL}. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7411–7428, 2024.
 - [49] Zheang Huai, Keshab K Parhi, and Xinmiao Zhang. Efficient architecture for long integer modular multiplication over solinas prime. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 146–151. IEEE, 2021.

- [50] Sergey Ioffe. Improved consistent sampling, weighted minhash and L1 sketching. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 246–255. IEEE Computer Society, 2010.
- [51] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.
- [52] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [53] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976, 2012.
- [54] Daniel M Kane and Jelani Nelson. Sparsen johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.
- [55] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics.
- [56] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O’neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340, 2016.
- [57] Rohan Kumar. Simply unmatched, truly limitless: Announcing Azure Synapse Analytics. Microsoft Azure Blog, <https://azure.microsoft.com/en-us/blog/simply-unmatched-truly-limitless-announcing-azure-synapse-analytics/>, November 2019. Accessed August 19, 2026.
- [58] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, 1997.
- [59] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.
- [60] Feng Li, Jianfeng Ma, Yinbin Miao, Ximeng Liu, Jianting Ning, and Robert H. Deng. A survey on searchable symmetric encryption. *ACM Computing Surveys*, 56:1 – 42, 2023.

- [61] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2356–2362, New York, NY, USA, 2021. Association for Computing Machinery.
- [62] Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 753–783. Springer, 2022.
- [63] Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. Wwv'18 open challenge: Financial opinion mining and question answering. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, pages 1941–1942, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.
- [64] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.
- [65] Evangelia Anna Markatou and Roberto Tamassia. Full database reconstruction with access and search pattern leakage. In *International Conference on Information Security*, pages 25–43. Springer, 2019.
- [66] Jiri Matousek. On variants of the johnson–lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.
- [67] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296, 2018.
- [68] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW '11*, page 113–124, New York, NY, USA, 2011. Association for Computing Machinery.
- [69] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.
- [70] Muhammad Naveed. The fallacy of composition of oblivious RAM and searchable encryption. Cryptology ePrint Archive, Paper 2015/668, 2015.
- [71] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human generated machine reading comprehension dataset, November 2016.

- [72] Jianting Ning, Jia Xu, Kaitai Liang, Fan Zhang, and Ee-Chien Chang. Passive attacks against searchable encryption. *IEEE Transactions on Information Forensics and Security*, 14(3):789–802, 2018.
- [73] NIST. `trec_eval` github repository. https://github.com/usnistgov/trec_eval, Accessed 2026-02-01.
- [74] NVIDIA. H100 GPU (product specifications). <https://www.nvidia.com/en-us/data-center/h100/>, 2026. Accessed: 2026-02-04.
- [75] NVIDIA. NVIDIA A100 Tensor Core GPU (specifications). <https://www.nvidia.com/en-us/data-center/a100/>, 2026. Accessed: 2026-02-04.
- [76] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 127–142. USENIX Association, August 2021.
- [77] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4753–4771, 2024.
- [78] David Pouliot and Charles V Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1341–1352, 2016.
- [79] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.
- [80] Navid Rekasaz, Oleg Lesota, Markus Schedl, Jon Brassey, and Carsten Eickhoff. Tripclick: The log files of a large health web search engine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '21*, pages 2507–2513, New York, NY, USA, 2021. Association for Computing Machinery.
- [81] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Ring oram: Closing the gap between small and large client storage oblivious ram. *IACR Cryptol. ePrint Arch.*, 2014:997, 2014.
- [82] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009.
- [83] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted Execution Environment: What It is, and What It is Not . In *2015 IEEE Trustcom/BigDataSE/ISPA*, pages 57–64, Los Alamitos, CA, USA, August 2015. IEEE Computer Society.

- [84] Darsh Shah, Tao Lei, Alessandro Moschitti, Salvatore Romeo, and Preslav Nakov. Adversarial domain adaptation for duplicate question detection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1056–1063, Brussels, Belgium, October–November 2018. Association for Computational Linguistics.
- [85] Victor Shoup. Arithmetic software libraries. In Joppe W. Bos and Martijn Stam, editors, *Computational Cryptography: Algorithmic Aspects of Cryptology*, pages 251–292. Cambridge University Press, 2021.
- [86] Victor Shoup. NTL: A library for doing number theory. <https://libntl.org>, 2021.
- [87] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*, pages 44–55. IEEE, 2000.
- [88] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.
- [89] Emil Stefanov, Marten van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. *Journal of the ACM (JACM)*, 65(4):1–26, 2018.
- [90] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *Proceedings 2014 Network and Distributed System Security Symposium*. Internet Society, 2014.
- [91] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.
- [92] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.
- [93] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing*, 41(2):293–331, 2012.
- [94] Vinod Vaikuntanathan and Or Zamir. Improving algorithmic efficiency using cryptography. *arXiv preprint arXiv:2502.13065*, 2025.
- [95] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

- [96] Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. Trec-covid: Constructing a pandemic information retrieval test collection. *SIGIR Forum*, 54(1), February 2021.
- [97] Henning Wachsmuth, Shahbaz Syed, and Benno Stein. Retrieval of the best counterargument without prior topic knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 241–251. Association for Computational Linguistics, 2018.
- [98] David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7534–7550, Online, November 2020. Association for Computational Linguistics.
- [99] Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, Ruicheng Yin, Changze Lv, Xiaoqing Zheng, and Xuanjing Huang. Searching for best practices in retrieval-augmented generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17716–17736, Miami, Florida, USA, November 2024. Association for Computational Linguistics.
- [100] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, October–November 2018. Association for Computational Linguistics.
- [101] Jinhao Zhu, Liana Patel, Matei Zaharia, and Raluca Ada Popa. Compass: encrypted semantic search with high accuracy. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 915–938, 2025.

A Details on BM25 Scoring

Recall that lexical search operates over a fixed vocabulary $\mathcal{V}$ of searchable terms. A query $Q \subseteq \mathcal{V}$ consists of a small subset of terms, and the goal is to identify the most relevant documents from a corpus $\mathcal{D}$ based on these terms. It is natural to implement lexical search by encoding queries and documents as sparse vectors. There are many possible instantiations of this encoding step. We use the BM25 ranking function [82], an extension of the TF-IDF framework [88] (Term Frequency - Inverse Document Frequency), because it is the most common in practice. However, the ideas here apply to nearly any lexical scoring metric.

The TF-IDF framework assigns each (keyword, document) pair a score indicating how relevant the keyword is to that document *relative to the other documents in the corpus*. The intuition is that uncommon words are more indicative of relevance when they appear in a query.

Let $\mathcal{D}$ be a corpus of text documents, let $\mathcal{V}$ be the vocabulary of keywords that can be searched for, and denote the number of possible keywords by $V = |\mathcal{V}|$. In TF-IDF, each keyword $q \in \mathcal{V}$ in a document $D \in \mathcal{D}$ gets a score of the form

$$\text{score}(q, D) = \text{tf}_{\mathcal{D}}(q, D) \cdot \text{idf}_{\mathcal{D}}(q), \quad (\text{A.1})$$

where the term frequency $\text{tf}_{\mathcal{D}}(q, D)$ increases with the prevalence of q in D , while the inverse document frequency $\text{idf}_{\mathcal{D}}(q)$ decreases with the prevalence of q in $\mathcal{D}$ as a whole.

This is equivalent to a sparse vector representation in which Q and D are encoded as bag-of-words vectors $\mathbf{v}_D, \mathbf{u}_Q \in \mathbb{R}^V$. The entries in $\mathbf{v}_D$ are weighted by their respective TF-IDF values, while $\mathbf{u}_Q$ is unweighted. With this encoding, the relevance can be computed as an inner product, $\text{rel}(Q, D) = \langle \mathbf{u}_Q, \mathbf{v}_D \rangle$.

BM25 is an instantiation of TF-IDF with

$$\text{tf}_{\mathcal{D}}(q, D) = \frac{\#(q, D) \cdot (k_1 + 1)}{\#(q, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|\mathcal{D}|}{\text{avg}(\mathcal{D})}\right)} \quad (\text{A.2})$$

and

$$\text{idf}_{\mathcal{D}}(q) = \ln \left(\frac{|\mathcal{D}| - \#_{\mathcal{D}}(q) + 0.5}{\#_{\mathcal{D}}(q) + 0.5} + 1 \right), \quad (\text{A.3})$$

where:

- $\#(q, D)$ is the number of occurrences of q in D ;
- $\#_{\mathcal{D}}(q)$ is the number of documents in $\mathcal{D}$ that contain q ;
- $\text{avg}(\mathcal{D})$ is the average document length in $\mathcal{D}$; and
- k_1, b are tunable parameters, set by default to $k_1 \approx 1.2$ and $b = 0.75$, but they can also be optimized for each corpus.

For a fixed corpus $\mathcal{D}$, vocabulary $\mathcal{V}$, and term $q \in \mathcal{V}$, the quantities $|\mathcal{D}|$, $\#_{\mathcal{D}}(q)$, $\text{avg}(\mathcal{D})$, k_1 , and b are fixed, and therefore so is $\text{idf}_{\mathcal{D}}(q)$. The score of q for D therefore depends only on the number of occurrences of q in D (and it is zero if q does not appear in D).

B Formal Metric Definitions

We formally define the three information retrieval metrics used in our evaluation:

Mean Reciprocal Rank (MRR@k). For a query q with a set of relevant documents $\mathcal{R}_q$, let r_q denote the rank of the first relevant document in the retrieved list (or ∞ if no relevant document appears in the top k positions). The reciprocal rank is:

$$\text{RR}(q) = \begin{cases} \frac{1}{r_q} & \text{if } r_q \leq k \\ 0 & \text{otherwise} \end{cases}$$

The Mean Reciprocal Rank over a set of queries $\mathcal{Q}$ is:

$$\text{MRR@k} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \text{RR}(q)$$

MRR emphasizes the position of the first relevant result, making it particularly suitable for question-answering tasks where users typically seek a single correct answer.

Recall@k. For a query q with relevant documents $\mathcal{R}_q$, let $\mathcal{D}_q^k$ denote the set of top k retrieved documents. Recall@k is defined as:

$$\text{Recall@k}(q) = \frac{|\mathcal{R}_q \cap \mathcal{D}_q^k|}{|\mathcal{R}_q|}$$

The average Recall@k over all queries is:

$$\text{Recall@k} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \text{Recall@k}(q)$$

Recall measures the fraction of relevant documents that appear in the top k results, emphasizing coverage over ranking quality.

Normalized Discounted Cumulative Gain (NDCG@k). NDCG is a graded relevance metric that accounts for both the position and relevance score of retrieved documents. For a query q , let $\text{rel}(d)$ denote the relevance score of document d (typically binary: 1 for relevant, 0 for non-relevant, though some datasets provide graded relevance). The Discounted Cumulative Gain at position k is:

$$\text{DCG@k}(q) = \sum_{i=1}^k \frac{\text{rel}(d_i)}{\log_2(i+1)}$$

where d_i is the document at rank i in the retrieved list. The Ideal DCG (IDCG) is the DCG of the perfect ranking:

$$\text{IDCG@k}(q) = \sum_{i=1}^{\min(k, |\mathcal{R}_q|)} \frac{\text{rel}(d_i^*)}{\log_2(i+1)}$$

where d_i^* are the documents sorted by decreasing relevance. The Normalized DCG is:

$$\text{NDCG@k}(q) = \frac{\text{DCG@k}(q)}{\text{IDCG@k}(q)}$$

The average NDCG@k over all queries is:

$$\text{NDCG@k} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \text{NDCG@k}(q)$$

NDCG provides a more nuanced evaluation than MRR or Recall, as it considers the entire ranking and applies logarithmic discounting to lower-ranked positions.

C Cryptographic Definitions and Proofs

We will prove the security of the cryptographic components from our lexical search protocol under the 1D-SLSN assumption of [4]. Note that in the exact version, where the client downloads all n encrypted scores from the server, the security follows directly from the security of EMVP (Lemma 4.2 of [4]).

C.1 The 1D-SLSN Assumption

Roughly, the 1D-Split Learning-Subspace-with-Noise (1DSLSN) assumption of [4] states that perturbed random codewords from a secret $[n, k]$ -linear code C are computationally indistinguishable from uniformly random vectors. They use the following notion of perturbation.

Definition C.1 (1D-SLSN). *Fix a finite field $\mathbb{F}$, a linear $[n, k]$ -code $C \subset \mathbb{F}^n$, and a split parameter s that divides n .² The 1DSLSN perturbation process for these parameters, denoted $1DSLSN[C, s](\mathbf{u})$, takes as input an arbitrary vector $\mathbf{u} \in \mathbb{F}^n$ and outputs the following distribution:*

1. Choose a uniform $\mathbf{c} \leftarrow C$ and set $\mathbf{u}' = \mathbf{u} + \mathbf{c}$;
2. Break $\mathbf{u}'$ into s equal-size blocks, $\mathbf{u}' = (\mathbf{u}'_1 | \mathbf{u}'_2 | \dots | \mathbf{u}'_s)$;
3. Choose s random invertible scalars $\alpha_1, \dots, \alpha_s \leftarrow \mathbb{F}^*$;
4. Output $\mathbf{v} = (\alpha_1^{-1} \mathbf{u}'_1 | \alpha_2^{-1} \mathbf{u}'_2 | \dots | \alpha_s^{-1} \mathbf{u}'_s) \in \mathbb{F}^n$.

Definition C.2 (1DSLSN Assumption [4], Def 3.3). *Fix a finite field $\mathbb{F}$, a family $\mathcal{C}$ of $[n, k]$ -linear codes over $\mathbb{F}$, and a split parameter $s \mid n$. The 1DSLSN assumption for these parameters asserts that no feasible adversary can win the following game with probability noticeably larger than $1/2$:*

1. A random code and a random bit are chosen: $C \leftarrow \mathcal{C}$, $\sigma \leftarrow \{0, 1\}$;
2. The adversary adaptively chooses $\mathbf{u}_1, \mathbf{u}_2, \dots \in \mathbb{F}^n$ and gets back for each one a sample $\mathbf{v}_i$, chosen according to:

$$\mathbf{v}_i \leftarrow \begin{cases} 1DSLSN[C, s](\mathbf{u}_i) & \text{if } \sigma = 1 \\ \text{Uniform in } \mathbb{F}^n & \text{otherwise;} \end{cases}$$

3. The adversary halts with a guess σ' ; it wins if $\sigma' = \sigma$.

C.2 Security Proofs

Lemma 4.2. [EMVP² Security] *The protocol in Fig. 4 is a semantically secure bilinear form protocol under the security of the PRF used for derivation from the secret key, the TDM construction, and the 1D-SLSN assumption of [4].* $\square$

Proof. We proceed by a hybrid argument. Let H_0 be the hybrid obtained by replacing the PRFs used to generate C and $\mathbf{Z}$ with uniformly random functions. Due to the security of these PRFs, it is sufficient to prove the security of H_0 . Now consider hybrid versions of the protocol H_1, H_2, H_3 as follows.

- H_1 is obtained by replacing the TDM $\mathbf{Z}$ in H_0 by a uniformly random $n \times n$ matrix.
- H_2 is obtained by modifying $\text{Enc}(\text{sk}, \mathbf{M})$ in H_1 to return a uniformly random $n \times n$ matrix.
- H_3 is obtained by modifying $\text{Enc}(\text{sk}_1, \mathbf{v})$ in H_2 to return a uniformly random vector in $\mathbb{F}^n$, in place of $\hat{\mathbf{v}}$.

²The definition extends easily to $s < n$ that does not divide n , but the phrasing becomes a little awkward.

Clearly, everything that the adversary sees in H_3 is random and independent of the data and queries, so we only need to establish that H_3 is indistinguishable from H_0 .

First, from the security of the TDM construction, the server’s views in H_0 and H_1 are equivalent. Next, for a uniformly random $\mathbf{Z}$ we have $\mathbf{A} + \mathbf{Z} \approx \mathbf{Z}$ for any matrix $\mathbf{A}$, so the adversary’s view is the same in H_1, H_2 . Finally, by the 1DSLSN assumption, the adversary cannot distinguish H_2 from H_3 . $\square$

Corollary C.3. *Let $\mathbf{M}_i$ be the matrix of sketched document vectors of the i th group and let $\mathbf{q}$ be the sketched query, as in Section 5. The protocol described in Section 4.1 for computing the squared norm $\|\mathbf{q}\mathbf{M}_i^T\|_2^2$ is secure.*

Proof. This can be viewed as an instance of the EMVP² protocol, where the encrypted matrix is $\mathbf{B}_i = \mathbf{M}_i^T \mathbf{M}_i$ and each query requests $\mathbf{w}\mathbf{B}_i\mathbf{v}^T$ for $\mathbf{w} = \mathbf{v} = \mathbf{q}$. $\square$

Corollary C.4. *The protocol for retrieving all the scores in the j th group is secure and private, i.e., it leaks no information about j to the server.*

D Additional Evaluation Details

D.1 Implementation Details

We used pyserini [61] to pre-process and tokenized all raw datasets uniformly using the default settings in pyserini. We note that our values in Table 5 for MS MARCO and TripClick differ from those reported in Table 2 of Compass [101] and [31]. We believe that the difference in corpus sizes are due to Compass using a different (unknown) pre-processing that likely dropped some documents. Additionally, their table reports the total number of raw queries whereas, we report the queries that are scored (have ground truth qrels). Compass and its baselines use the same scoring criteria, based on our understanding of their implementation. These changes were likely inherited by [31]. Neither difference is material to our measurements, especially since we ran the Inv-ORAM pipeline from scratch by pre-processing and tokenizing the same way as LEXICURE.

We implemented the LEXICURE client in C++, and used the square trapdoored-matrix construction described in section 5.2 of [4]. The client computation was measured single-threaded on a machine with a 64-core AMD EPYC 7R13 processor. For all experiments, we used $d = 10,000$ as the CountSketch output dimension and the standard BM25 parameters $k_1 = 1.2$ and $b = 0.75$. The underlying EMVP parameters target 128-bit security and use the overhead- $f = 4$ setting from [4]: $\ell = d = 10,000$, $k = 30,000$, $n = 40,000$, and block size $b = 2728$, giving $s = 15$. For LEXICURE-2 experiments on datasets with more than one million documents, we use $G = 10,000$ groups and vary K_G , the number of groups retrieved in round two. Figure 12 shows the MRR@10 tradeoff, while Figs. 16 and 17 show Recall@10 and NDCG@10, respectively.

Our implementation used the C++ NTL library [86] for computing matrix multiplication over fields. The floating-point matrix entries are mapped to field elements by truncating to a fixed number of bits and scaling up to an integer. The final scores are scaled down appropriately after decryption by the client. The scores fit comfortably in a 30-bit signed integer even after scaling. We use $2^{31} - 1$ as the prime modulus, with each field element occupying 4 bytes, but note that better NTL-friendly or FFT-friendly alternatives may exist [85, 43]. We use this representation for LEXICURE-1 and for the second round of LEXICURE-2. The first round of LEXICURE-2 is a quadratic form, so its group scores $\gamma_i = \|\mathbf{q}\mathbf{M}_i^T\|_2^2$ square the document scores and need a wider

modulus; we therefore account for the first-round matrices and results at 8 bytes per field element. We use AES-128 in counter mode for efficient and cryptographically secure pseudorandom function (PRF) random number generation, as needed for EMVP.

D.2 Sources of Accuracy Loss

The search quality of LEXICURE can be affected by three approximations that it makes: First, CountSketch compresses the sparse BM25 vectors to dimension $d = 10,000$, while preserving the inner products approximately (but not completely); this is the main source of accuracy loss in LEXICURE-1. Second, LEXICURE-2 searches only documents in the selected K_G groups and may therefore omit relevant documents; this additional loss depends on K_G and the corpus ordering. Finally, encoding the similarity scores in a fixed-precision field (as needed for EMVP) introduces rounding error, but we expect this error to be very small with our 30-bit representation. Note that the EMVP protocol itself computes exactly over the field, so there is no additional error there.

D.3 Additional Lexicure-1 Results

D.3.1 Measured Client Computation

Our single-threaded client instance required up to 24 MB of RAM. All measurements represent single-query client computation without parallelization. The per-query client computation for various corpus sizes is shown in Fig. 13. For TripClick (1.5M documents) and MS MARCO (8.8M documents), representing the lower and upper end of the corpora with more than one million documents, the measured client times are 0.78 seconds and 4.38 seconds, respectively. We present the breakdown in Fig. 14. Decryption dominates client computation and scales linearly with the number of documents. Encryption, which depends only on the CountSketch output dimension, takes less than 10 ms.

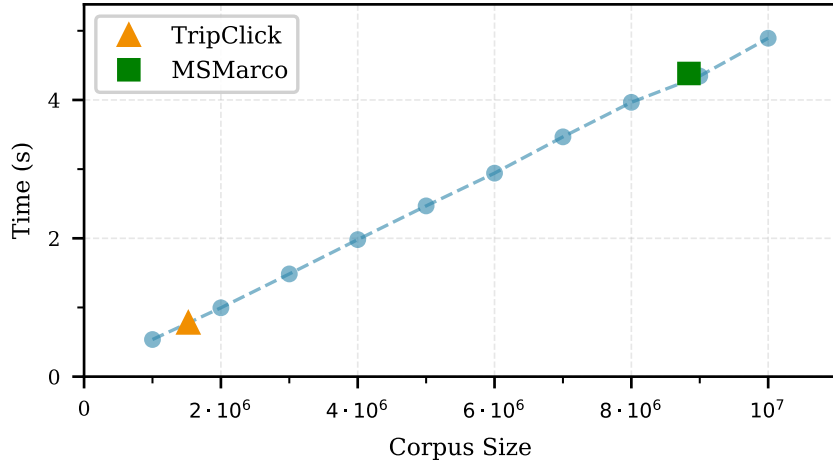


Figure 13: Measured client-side computation time, including encryption, decryption, and result processing, as a function of dataset size.

The primary bottleneck within decryption is evaluating the trapdoored-matrix (TDM) contribution. Across all measured corpus sizes, TDM evaluation accounts for over 95% of decryption

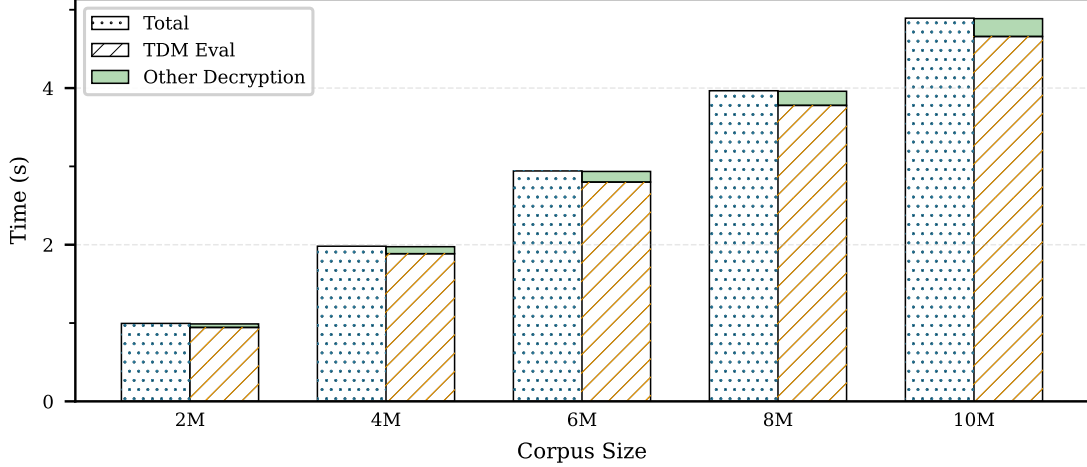


Figure 14: Breakdown of measured client-side computation time. Encryption is included in the total and is small relative to decryption, which is dominated by TDM evaluation.

time. Our implementation requires NTT operations for efficient evaluation [4]. There are several ways to reduce this client time in practice. First, it may be possible to switch to better TDM constructions, the rectangular TDMs discussed in Section 5.4 support linear-time multiplication and avoid NTT operations using cache-friendly linear codes [11, 35]. Also, each TDM block can be evaluated independently, making decryption parallelizable.³ The TDM computation also depends only on the encrypted query and can run while the client waits for the server result.

D.3.2 Projected Server Computation

As described in the main evaluation, the server computes s matrix-vector products on matrices with $1/s$ of the encrypted index columns. Together, these products require approximately $2Ndf$ modular operations over the EMVP field: Ndf multiplications and $N(df - 1)$ additions [4].

For MS MARCO with $N = 8.8\text{M}$ documents, $d = 10,000$, $n = 40,000$, and $s = 15$, the server performs approximately 707 billion modular operations per query. Modular reduction using a Solinas prime of the form $2^k - c$ replaces division with shifts and additions [49]. We estimate that this workload corresponds to roughly 1.4–3.5 TFLOP-equivalent computation, accounting for a 2–5 $\times$ overhead relative to floating-point fused multiply-add operations. Based only on peak arithmetic throughput, this computation could take as little as **12–15 ms** on an NVIDIA A100 or H100 [75, 74]. This projection excludes data movement, and the server will be memory-bandwidth-bound in many settings because each query scans the encrypted index. Sharding the index across accelerators can increase aggregate memory bandwidth.

D.4 Accuracy at a Rank Cutoff of 100

The main text reports all quality metrics at a rank cutoff of $K = 10$ (Table 11). For completeness, Table 15 repeats the same comparison at a cutoff of $K = 100$, computed from the same run files.

³In contrast, parallelizing ORAM is less straightforward [17].

The relative accuracy of LEXICURE-1 with respect to plaintext BM25 is essentially unchanged by the cutoff.

Table 15: Evaluation Results: BM25 vs Lexicure measured at a top-100 cutoff.

Method ($\rightarrow$)	Recall@100			MRR@100			NDCG@100		
Dataset ($\downarrow$)	BM25	Lexicure	%	BM25	Lexicure	%	BM25	Lexicure	%
ArguAna	0.932	0.923	99.0%	0.207	0.203	97.6%	0.365	0.359	98.3%
Climate-FEVER	0.425	0.415	97.6%	0.232	0.228	97.9%	0.225	0.218	97.2%
CQADupStack	0.528	0.510	96.6%	0.288	0.274	95.2%	0.321	0.307	95.6%
DBPedia	0.468	0.457	97.7%	0.589	0.552	93.7%	0.373	0.360	96.6%
FEVER	0.918	0.911	99.2%	0.628	0.615	97.9%	0.678	0.665	98.2%
FiQA	0.539	0.522	96.7%	0.305	0.290	94.9%	0.299	0.286	95.6%
HotpotQA	0.796	0.788	99.0%	0.806	0.795	98.6%	0.668	0.657	98.3%
MSMARCO	0.658	0.649	98.7%	0.195	0.191	98.0%	0.287	0.282	98.3%
NFCorpus	0.246	0.247	100.5%	0.528	0.529	100.3%	0.274	0.273	99.5%
NQ	0.751	0.735	97.8%	0.275	0.260	94.8%	0.367	0.352	96.0%
Quora	0.973	0.973	100.0%	0.782	0.780	99.7%	0.811	0.809	99.7%
SCIDOCS	0.348	0.334	96.1%	0.272	0.264	97.1%	0.213	0.206	96.8%
SciFact	0.925	0.913	98.7%	0.650	0.638	98.1%	0.707	0.693	98.1%
Touche-2020	0.582	0.542	93.0%	0.744	0.646	86.9%	0.520	0.456	87.7%
TREC-COVID	0.109	0.100	91.7%	0.853	0.834	97.8%	0.438	0.400	91.4%
TripClick	0.499	0.470	94.1%	0.289	0.261	90.0%	0.263	0.243	92.6%
Avg. performance	97.3%			96.2%			96.2%		

D.5 Additional Lexicure-2 Results

The main evaluation reports the MRR@10 tradeoff obtained by fixing $G = 10,000$ and varying K_G on datasets with more than one million documents. Here we report the corresponding Recall@10 and NDCG@10 results, followed by additional server-storage, computation, and locality analyses.

D.5.1 Additional Accuracy Metrics

Figures 16 and 17 show the bandwidth–accuracy tradeoffs using MRR@10 and NDCG@10, respectively. These plots complement the Recall@10 results in Fig. 12.

All three metrics follow the same general trend: MS MARCO, NQ, and TripClick have better accuracy–bandwidth tradeoffs than Climate-FEVER, DBPedia, and FEVER across all three metrics.

D.5.2 Bandwidth and Server Storage

The choice of K_G affects bandwidth but not server storage. We keep $G = 10,000$ and $d = 10,000$ fixed because lowering either substantially reduced accuracy. With G and d fixed, we vary the EMVP parameter sets used for the document and group dimensions. These choices trade client bandwidth for server storage without affecting accuracy or security guarantees. All evaluated settings target 128-bit security. Figure 18 shows this tradeoff for a corpus the size of MS MARCO (8.8M documents).

As can be seen in Fig. 18, lower overhead factors reduce server storage but increase bandwidth due to less efficient encoding. The “Balanced Tradeoff” points (green stars) identify the configuration minimizing the product of server storage and client communication, which needs no threshold parameter and is insensitive to the units of either axis. For both $K_G = 10$ and $K_G = 100$ this occurs at $f_{\text{ndim}} = 2$ and $f_{\text{ngroups}} = 4$, achieving 17.13 TiB of server storage with 41.63 MiB and 173.85

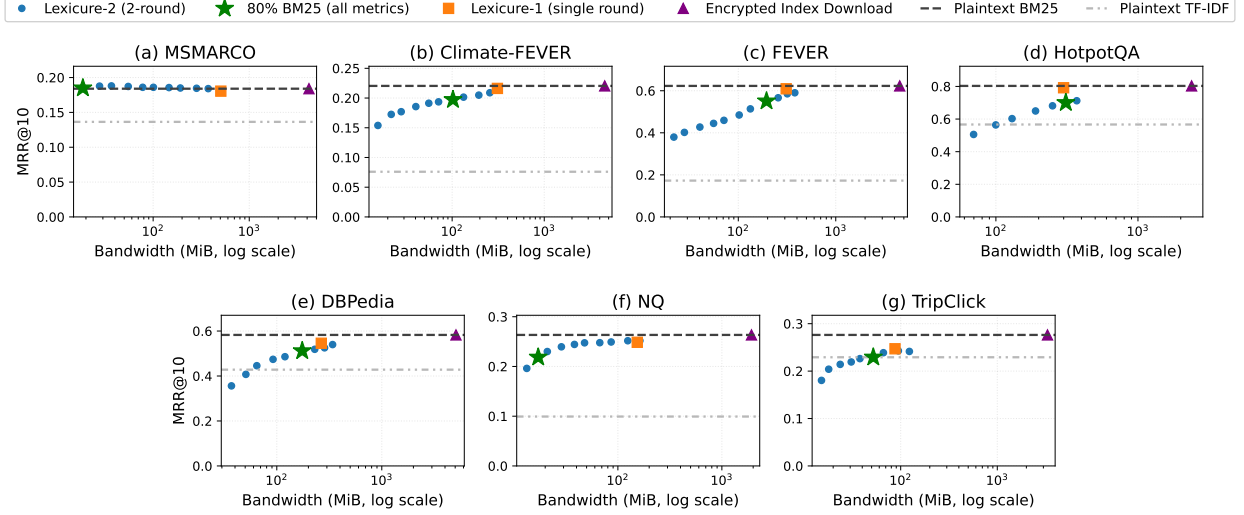


Figure 16: MRR@10 vs. bandwidth for LEXICURE-2 on datasets with more than one million documents. Each curve varies K_G , and the markers have the same meaning as in Fig. 12. Datasets are listed in decreasing order of index size.

MiB of communication, respectively. The two-round protocol enables a range of storage-bandwidth tradeoffs not achievable with the single-round protocol.

D.5.3 Client and Server Computation

The server computation for LEXICURE-2 consists of two rounds. Round one computes G group scores, where each group requires a quadratic form $\mathbf{q}^T \mathbf{M}_i \mathbf{q}$ with $\mathbf{M}_i$ being a $(d \cdot f_{\text{ndim}}) \times (d \cdot f_{\text{ndim}})$ matrix. Round two computes individual document scores for the K_G selected groups using $(d \cdot f_{\text{ndim}})$ -dimensional vectors. For the balanced configuration identified in Fig. 18, $f_{\text{ndim}} = 2$ and $f_{\text{ngroups}} = 4$, round one requires 8 trillion modular operations; with $K_G = 10$ round two adds 0.35 billion, for $11.3\times$ the 707 billion operations required by LEXICURE-1 with $f = 4$. With $K_G = 100$ the round-two cost rises to 3.5 billion, leaving the total at 8 trillion operations, still $11.3\times$ LEXICURE-1. Using Solinas prime optimization ($2\text{--}5\times$ overhead), this translates to 16–40 TFLOPs. Based only on peak arithmetic throughput, the corresponding compute time is 51–128 ms on an NVIDIA A100 GPU (312 TFLOPS FP16). These projections exclude data movement and may be memory-bandwidth-bound.

The client decrypts G group scores in round one and $K_G N/G$ document scores in round two. As in LEXICURE-1, TDM evaluation is the main client-side cost. For the standard setting, where $d = G$ and both dimensions use the same encoded length n , the rectangular-TDM optimization from Section 5.4 applies to the round-two masks, reducing this work from $O((G + K_G N/G)n \log n)$ to $O(Gn \log n + K_G(N/G)n)$. Round one keeps its NTTs, since each first-round matrix is square. The TDM operations can also be parallelized across cores, and their query-dependent portions can run while the client waits for server responses. LEXICURE-2 retains the minimal persistent client state of LEXICURE-1, consisting only of cryptographic keys and the CountSketch seed.

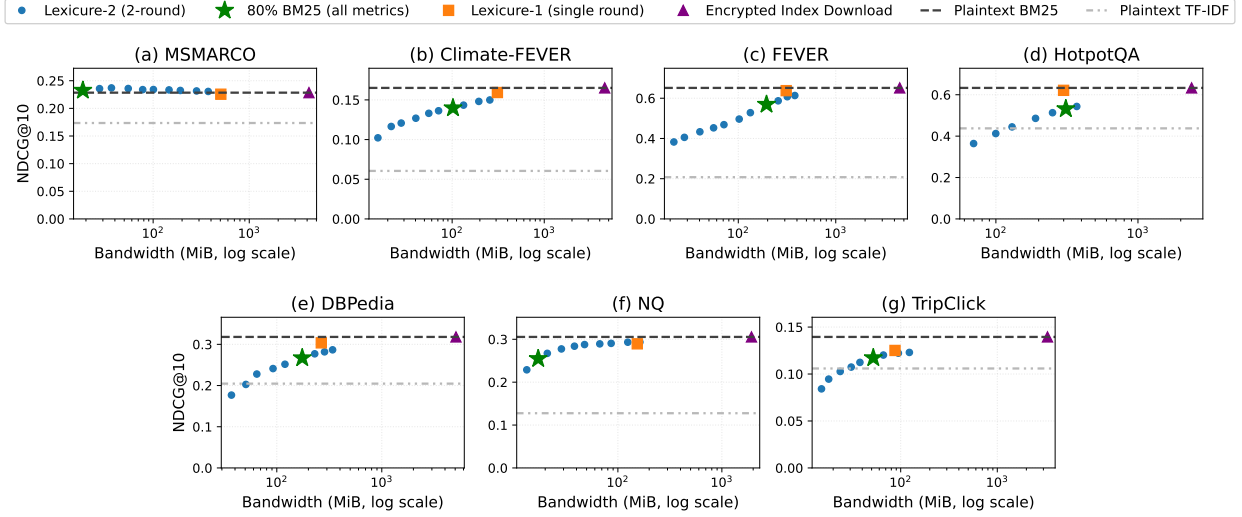


Figure 17: NDCG@10 vs. bandwidth for LEXICURE-2 on datasets with more than one million documents. Each curve varies K_G , and the markers have the same meaning as in Fig. 12. Datasets are listed in decreasing order of index size.

D.5.4 Locality Analysis

To better understand the dataset-dependent performance of LEXICURE-2, we conducted a post-hoc analysis examining whether document locality in the corpus ordering distributed with each dataset might explain the observed differences. This analysis is preliminary and exploratory in nature.

We heuristically measure locality by computing the cosine similarity of document pairs separated by index distance r under that ordering. The locality score is a weighted average with weights proportional to $1/r$ to emphasize similarity among nearby documents. This ordering is the same as the one used to partition documents into groups for LEXICURE-2.

Figure 19 shows the locality scores for the large datasets in our evaluation. NQ and MS MARCO exhibit the highest locality scores (0.17–0.26), followed by TripClick (0.13), while HotpotQA and DBPedia are lower (0.07) and Climate-FEVER and FEVER lower still (0.02), with similarity profiles that are flat across all distances.

We observe a correlation between these locality scores and the accuracy–bandwidth tradeoff of LEXICURE-2: datasets with higher locality tend to maintain better accuracy at lower bandwidth (see Fig. 12). However, this is a preliminary observation based on the default corpus orderings provided with these datasets and a potential direction for future work.

E Alternative Sketching Methods

Recall that we need to reduce the dimension of the sparse BM25 document and query vectors while preserving their inner products. We considered several options for this densification step, with the goal of maintaining retrieval quality and minimizing the dimension. Below we summarize the approaches that we explored and why we ultimately retained CountSketch [18].

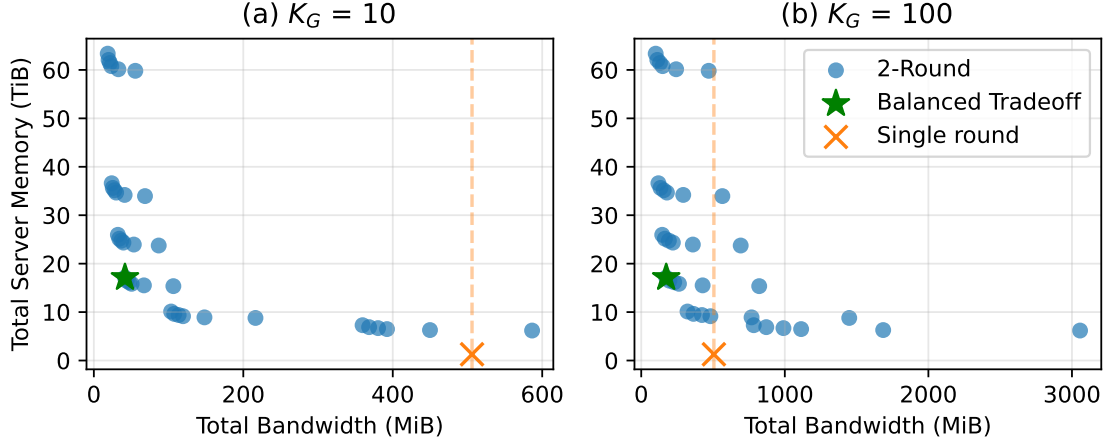


Figure 18: Server storage vs. client bandwidth for LEXICURE-2 on a corpus the size of MS MARCO (8.8M documents) with different EMVP² parameter settings. (a) $K_G = 10$ groups retrieved. (b) $K_G = 100$ groups retrieved. Each point represents a different combination of f_{ndim} and f_{ngroups} . The green star (“Balanced Tradeoff”) marks the configuration that minimizes the product of server storage and client communication. The orange cross (Single round) is LEXICURE-1. The two-round protocol (blue dots) offers storage–bandwidth tradeoffs not achievable with the single-round protocol.

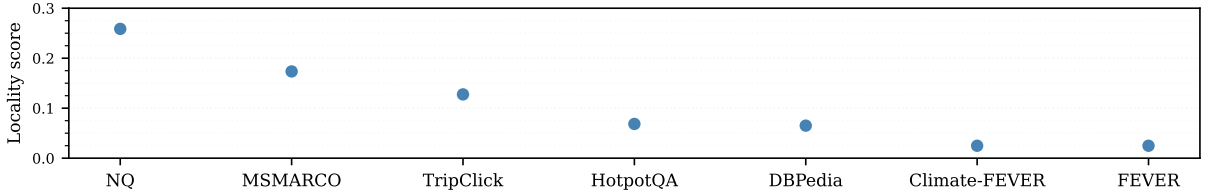


Figure 19: Locality scores for large-scale datasets. The locality score is a $1/r$ -weighted average of the cosine similarity between documents r apart in the corpus ordering. Higher scores indicate that the corpus ordering distributed with the dataset places similar documents near each other.

Linear projections. Linear hashing such as CountSketch or dense Gaussian based on Johnson–Lindenstrauss transforms, all preserve inner products of $\mathbf{a} \cdot \mathbf{b}$ with error on the order of $\|\mathbf{a}\|_2 \|\mathbf{b}\|_2 / \sqrt{d}$; they mainly differ in constants and construction cost [66, 54]. CountSketch is attractive due to its simplicity and speed, but in our context it is only used during index construction so reducing the dimension is a lot more important than computation speed. We had hoped that a dense projection will perform better (since it spreads every term over every output coordinate), and tested this hypothesis using Gaussian projections on several BEIR datasets. Our experiments indicated that across the spectrum, Gaussian projection was not any better than CountSketch. For example, on FiQA, 10,000-dimensional CountSketch retained 95–97% of plaintext BM25 quality, depending on the metric, whereas Gaussian projection retained only 93–94% at dimension 8,192 and 88–89% at dimension 5,000.⁴

⁴These experiments used floating-point projections and thus omit quantization errors that would only further degrade quality.

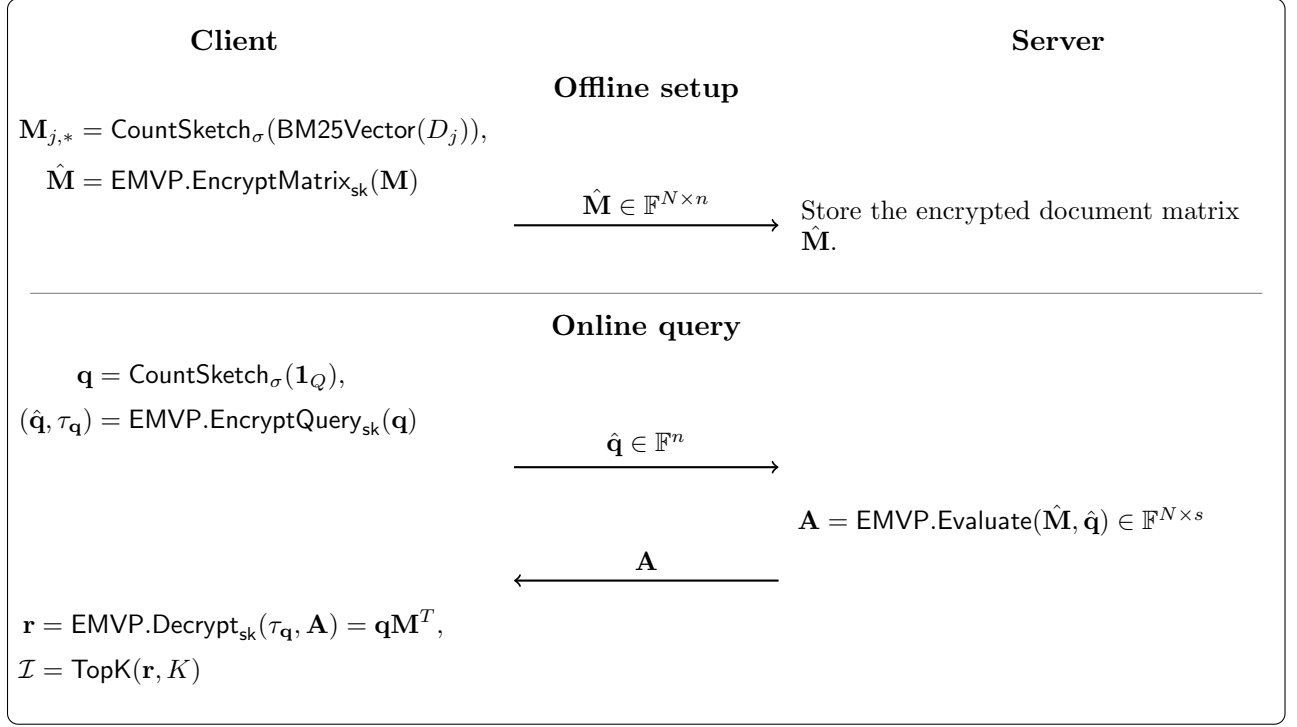


Figure 20: LEXICURE-1 message flow. The offline upload is performed once; each query uses one client-to-server message and one server-to-client response.

Separating high-weight terms. We also considered assigning dedicated coordinates to the highest-IDF terms and sketching only the remaining vocabulary. This would help if a small set of terms accounted for most of the squared BM25 weight. In natural-language corpora, however, many terms occur only once and consequently receive the same near-maximum IDF. The upper tail is therefore nearly flat rather than concentrated. In our diagnostics, the 1,000 largest IDF values accounted for only 5.3% of squared IDF mass on ArguAna and 0.6% on TREC-COVID. Reserving coordinates for these terms would consume substantial dimension while barely reducing the residual sketching error.

Sampling-based sketches. Priority sampling, threshold sampling, and weighted MinHash can improve inner product estimates in some regimes by exploiting support intersections rather than preserving all pairwise inner products [28, 50]. They do not, however, directly produce a fixed dense linear representation whose ordinary inner product is the BM25 score.

Choice of densifier. Our results do not imply that CountSketch is universally optimal. They show that, for the sparse BM25 vectors and EMVP interface considered here, the alternatives we evaluated offer no useful reduction below dimension 10,000. We therefore chose CountSketch: it achieves retrieval quality as good as any alternative we tried, has the lowest construction cost, and integrates directly with the encrypted matrix-vector computation.

Algorithm 2 LEXICURE-2: 2-round encrypted lexical search

Offline setup (Params: prime p , integers d, G):

Client Input: Vocabulary $\mathcal{V}$, Documents $\mathcal{D} = (D_1, \dots, D_N)$

- 1: $\text{sk} \leftarrow \text{EMVP2.KEYGEN}$, $\sigma \leftarrow \text{Sketching-seed}$
- 2: $\mathbf{M} \leftarrow \text{Sketch}_\sigma(\text{BM25}(\mathcal{D})) \in \mathbb{F}_p^{N \times d}$; $L \leftarrow N/G$
- 3: Partition $\mathbf{M} = (\mathbf{M}_1; \dots; \mathbf{M}_G)$, $\mathbf{M}_i \in \mathbb{F}_p^{L \times d}$
- 4: $\mathbf{B}_i \leftarrow \mathbf{M}_i^T \mathbf{M}_i$ for all $i \in [G]$
- 5: $\mathbf{P}_j \leftarrow ((\mathbf{M}_1)_{j,*}; \dots; (\mathbf{M}_G)_{j,*}) \in \mathbb{F}_p^{G \times d}$ // regroup rows
- 6: $\hat{\mathbf{B}}_i \leftarrow \text{EMVP2.ENCRYPTMATRIX}(\text{sk}, \mathbf{B}_i)$ for all $i \in [G]$
- 7: $\hat{\mathbf{P}}_j \leftarrow \text{EMVP2.ENCRYPTMATRIX}(\text{sk}, \mathbf{P}_j)$ for all $j \in [L]$
- 8: Send $\{\hat{\mathbf{B}}_i\}_{i \in [G]}$ and $\{\hat{\mathbf{P}}_j\}_{j \in [L]}$ to the server

Online query

Client Input: $Q \subseteq \mathcal{V}$, candidate-group count K_G , result count K

Client:

- 9: $\mathbf{q} \leftarrow \text{Sketch}_\sigma(\mathbf{1}_Q)$
- 10: $(\hat{\mathbf{q}}, \tau_{\hat{\mathbf{q}}}) \leftarrow \text{EMVP2.ENCRYPTQUERY}(\text{sk}, \mathbf{q})$; send $\hat{\mathbf{q}}$

Server:

- 11: $\mathbf{A}_i \leftarrow \text{EMVP2.EVALUATE}(\hat{\mathbf{B}}_i, \hat{\mathbf{q}}, \tau_{\hat{\mathbf{q}}})$ for all $i \in [G]$
- 12: Send $\{\mathbf{A}_i\}_{i \in [G]}$ to the client

Client:

- 13: $\gamma_i \leftarrow \text{EMVP2.DECRYPT}(\text{sk}, \tau_{\hat{\mathbf{q}}}, \mathbf{A}_i) = \mathbf{q} \mathbf{B}_i \mathbf{q}^T$ for all $i \in [G]$
- 14: $\mathcal{I} \leftarrow \text{TOPK}((\gamma_1, \dots, \gamma_G), K_G)$ // top K_G groups
- 15: **for all** $i \in \mathcal{I}$ **do** // encrypt one-hot vectors
- 16: $(\hat{\mathbf{e}}_i, \tau_i) \leftarrow \text{EMVP2.ENCRYPTQUERY}(\text{sk}, \mathbf{e}_i)$
- 17: **end for**
- 18: Send $\{\hat{\mathbf{e}}_i\}_{i \in \mathcal{I}}$ to the server // $\mathcal{I}$ remains secret

Server:

- 19: **for all** $i \in \mathcal{I}$ **do**
- 20: $\mathbf{R}_{i,j} \leftarrow \text{EMVP2.EVALUATE}(\hat{\mathbf{P}}_j, \hat{\mathbf{e}}_i, \tau_i)$ for all $j \in [L]$
- 21: **end for**
- 22: Send $\{\mathbf{R}_{i,j}\}_{i \in \mathcal{I}, j \in [L]}$ to the client

Client: // Recover scores from the top K_G groups, $\mathbf{r}_i = \mathbf{q} \mathbf{M}_i^T$

- 23: $(\mathbf{r}_i)_j \leftarrow \text{EMVP2.DECRYPT}(\text{sk}, \tau_i, \tau_{\hat{\mathbf{q}}}, \mathbf{R}_{i,j}) \forall i \in \mathcal{I}, j \in [L]$
 - 24: **return** $\text{TOPK}(\bigcup_{i \in \mathcal{I}} \mathbf{r}_i, K)$
-

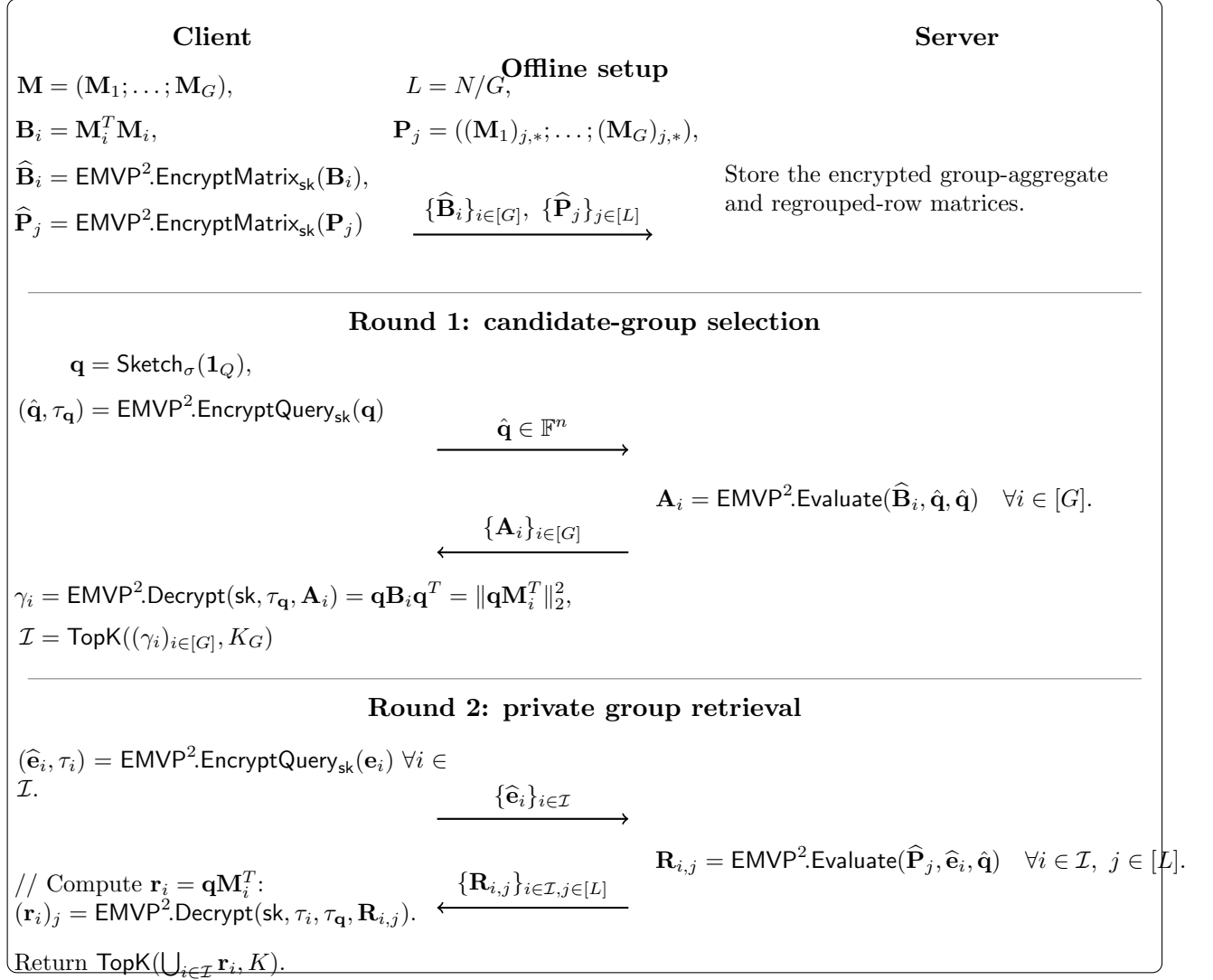
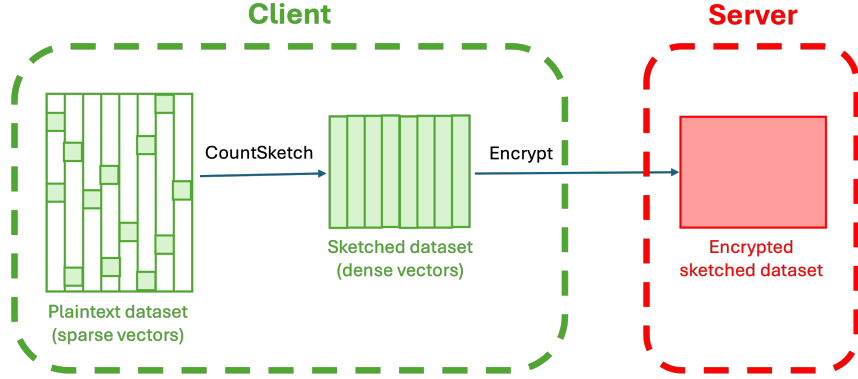
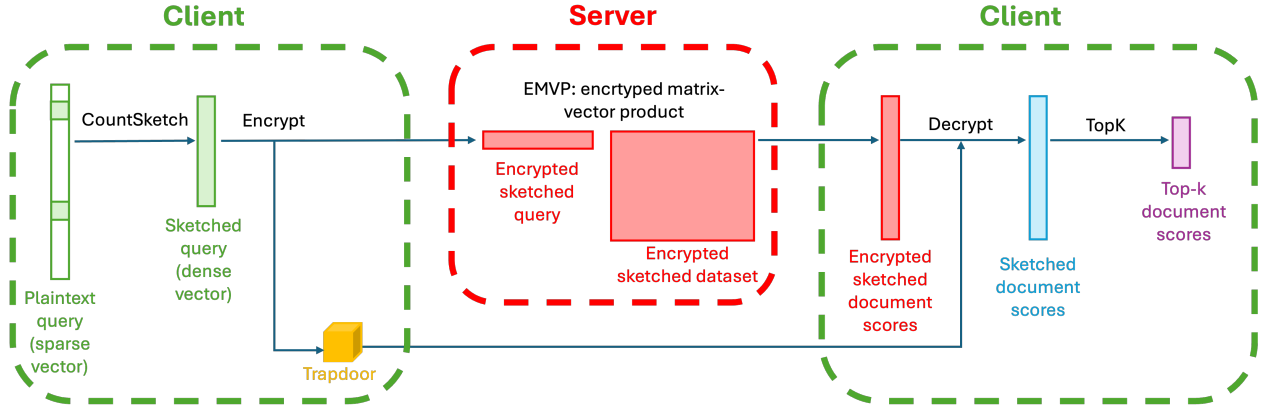


Figure 21: LEXICURE-2 message flow. The encrypted query $\hat{\mathbf{q}}$ is retained by the server between rounds. Encrypted one-hot selectors hide which K_G groups the client retrieves in the second round.



(a) LEXICURE-1 setup. The input is a dataset of documents, each represented as a sparse (column) vector with a coordinate per vocabulary terms. The non-zero entries contain the TF-IDF or BM25 scores of the terms appearing in the document.



(b) LEXICURE-1 query. The query is given as a sparse one-hot vector of the vocabulary terms appearing in the query. Exact TF-IDF or BM25 scores are obtained by a matrix-vector product of this query vector with the plaintext dataset matrix (the leftmost matrix in Fig. 22a). LEXICURE-1 replaces this with encrypted matrix-vector product (EMVP) over query and document vectors sketched with CountSketch. After the top-k document IDs are recovered, the corresponding documents are retrieved from the server with standard PIR.