CoSecRAG: Efficient Vector Similarity Search with Database and Query Privacy for RAG

Moyang Xie
Nanjing University
Nanjing, China
xie_moyang@foxmail.com

Jiacheng Gao
Nanjing University
Nanjing, China
jcgao@smail.nju.edu.cn

Yuan Zhang
Nanjing University
Nanjing, China
zhangyuan@nju.edu.cn

Sheng Zhong
Nanjing University
Nanjing, China
zhongsheng@nju.edu.cn

ABSTRACT

Retrieval-augmented generation (RAG) grounds large language models on external knowledge by using vector similarity search over document embeddings to identify relevant documents and supplying their contents as generation context. However, its retrieval stage creates privacy risks: user queries may reveal sensitive intents, while the database may contain proprietary or confidential records. Existing secret-sharing-based solutions that protect both query and database privacy remain costly. To make private vector similarity search more efficient, we present CoSecRAG, a two-server system for RAG that protects both database and query embeddings. CoSecRAG introduces Inner Product Preserving Query Mask (IPQ-Mask) to lower similarity-scoring costs by avoiding online secure multiplications and Post-Scoring Cluster Pruning (PSCP) to reorganize the execution pipeline for approximate retrieval, reducing private-extraction volume and secure top-K selection costs. We implement CoSecRAG and evaluate it on six BEIR datasets and large-scale synthetic benchmarks. CoSecRAG maintains strong retrieval quality while substantially reducing the cost of private vector similarity search. Compared with P^{2}RAG, the most recent two-server baseline for private vector similarity search in RAG, CoSecRAG achieves up to 181× online score-computation speedup and 81× online speedup in secure top-K computation. End-to-end, it achieves up to 99× online and 34× offline runtime speedups. Compared with PRAG, CoSecRAG achieves 19×–58× online and 2×–16× offline end-to-end speedups.

1 INTRODUCTION

Vector databases organize unstructured data as dense embeddings and support similarity search at scale [38]. This retrieval abstraction underlies applications such as semantic search, recommendation, and multimedia retrieval [48]. It has also become a key component of retrieval-augmented generation (RAG), where vector similarity search over document embeddings identifies relevant documents, whose contents are then supplied to a large language model (LLM) as external context [29]. By grounding generation in retrieved evidence, RAG helps mitigate LLM hallucinations [21] and enables LLMs to incorporate up-to-date, private, and domain-specific knowledge without model retraining, thereby improving factual grounding and domain adaptability [29]. Accordingly, RAG has been adopted in organizational applications such as enterprise customer service, finance, healthcare, and law [23, 36, 51], as well as in personal productivity assistants such as Microsoft 365 Copilot [33].

However, many practical RAG deployments outsource the retrieval of relevant documents, LLM inference, or both to remote services rather than execute the entire pipeline within a single trusted environment [7, 34, 57]. This division of computation crosses trust boundaries: users submit queries for remote processing, database owners delegate the retrieval of relevant documents from their knowledge bases, and model providers offer LLM inference over proprietary models [12]. Without appropriate privacy-preserving measures, outsourced processing may expose these assets to the executing services, forcing their owners to trust those services. The user must protect inputs and query embeddings that may reveal personal information, sensitive intentions, or private context; the database owner must protect proprietary or confidential documents and their embeddings; and the model provider must protect model parameters as valuable intellectual property. Protecting these assets securely and efficiently is therefore essential to realizing the full potential of RAG in practice.

Achieving this protection requires securing both components of a RAG pipeline: relevant-document retrieval, which identifies relevant documents in a knowledge base and obtains their contents, and LLM inference, which may process the user input before retrieval or consume the retrieved context afterward. A large body of work has studied secure LLM inference [1, 6, 10–12, 16, 30–32, 39]. Existing approaches to protecting the retrieval of relevant documents, however, still face important limitations.

Specifically, many existing approaches fail to simultaneously protect query and database privacy [5, 7, 13, 28, 35, 43, 44, 47, 50, 53, 55, 56]. Some other approaches rely on homomorphic encryption [2, 45, 56], which hinders their integration into the commonly used secret-sharing setting for secure LLM inference [11, 12, 16, 31, 32, 39]. Against this backdrop, recent work has explored how to use secret sharing to protect both query and database embeddings when retrieving relevant documents. PRAG [57] and P^{2}RAG [34] represent important advances in this direction. Their retrieval costs, however, remain substantial: in our end-to-end experiments with $N = 2^{22}$ documents, PRAG and P^{2}RAG incur online retrieval latencies of 595.58 s and 3,152.33 s, respectively.

One practical approach to securely retrieve relevant documents is to organize the retrieval process into two sequential steps: vector similarity search followed by document fetching, as reflected in

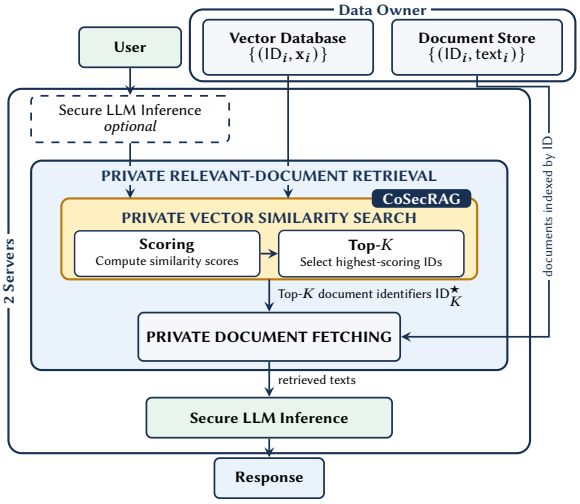


Figure 1: Privacy-preserving RAG system with CoSecRAG.

existing private RAG systems [2, 7, 34, 57]. In the first step, the system performs similarity search over a vector database of document embeddings. It takes as input a query embedding that may be generated directly from the user's input or obtained as the output of an upstream secure computation, e.g., secure LLM inference. The system then returns the identifiers of the documents with the most similar embeddings. In the second step, the returned identifiers are used to securely fetch the original document contents, e.g., via private information retrieval (PIR) [14]. The retrieved texts are then supplied as context to downstream secure LLM inference.

Focusing on the first step, we present CoSecRAG, an efficient vector similarity search system for RAG that protects both query and database embeddings. CoSecRAG operates under additive secret sharing between two servers and proceeds in two main steps. The scoring step computes similarity scores between the query and database embeddings, while the top-K selection step privately identifies the document identifiers associated with the K highest scores.

CoSecRAG is motivated by a key observation: protecting both query and database embeddings creates two key performance bottlenecks, one in scoring and the other in top-K selection.

In the scoring step, protecting both query and database embeddings means that both operands used to compute each inner-product similarity score must remain private. Consequently, their coordinate products require costly secure multiplications. This cost is evident in P^{2}RAG [34], which directly computes such scores for all documents and thus performs secure multiplications across all document-coordinate pairs.

In the top-K selection step, protecting both sides also requires the resulting scores and query-dependent access patterns to remain hidden. Directly selecting the document identifiers associated with the K highest scores therefore relies on secure selection over all private scores. P^{2}RAG follows this approach over entire-database scores, resulting in substantial cost. Pruning before scoring can reduce the selection space, but it requires the embeddings in the query-dependent selected clusters to be extracted without revealing either the selection or the database contents. For example, PRAG [57] uses cluster pruning [20], yet its private extraction still operates on high-dimensional document embeddings, incurring substantial cost.

Based on this observation, we design CoSecRAG to address both bottlenecks and make the following contributions:

- We introduce *Inner Product Preserving Query Mask* (IPQ-Mask) for secure scoring. Before the query arrives, IPQ-Mask precomputes the product of the database embedding matrix and a one-time random query mask. Online, the servers open only the difference between the query and the mask, and combine it with the precomputed product through local computation. This eliminates online multiplications between secret shares from scoring over the full database and substantially reduces online communication.
- We introduce *Post-Scoring Cluster Pruning* (PSCP) for secure top-K selection. PSCP first uses IPQ-Mask to compute document and cluster scores, and then selects clusters and extracts only scalar score records. By reversing the conventional prune-before-score pipeline, PSCP reduces the final secure selection from entire-database scores to a cluster-pruned candidate set and meanwhile avoids private extraction of high-dimensional document embeddings.
- We combine IPQ-Mask and PSCP in CoSecRAG, a complete protocol for privacy-preserving vector similarity search in RAG. We analyze its privacy under the non-colluding semi-honest model. The resulting protocol protects both query and database embeddings and hides intermediate retrieval results from each individual server.
- We implement CoSecRAG and evaluate its retrieval quality, the costs of its individual steps, and the end-to-end performance of private vector similarity search on six BEIR datasets and large-scale synthetic benchmarks. CoSecRAG maintains strong retrieval quality while substantially reducing the cost of private vector similarity search. Compared with P^{2}RAG, it achieves up to 181× online speedup in score computation and 81× online speedup in secure top-K computation. End-to-end, it achieves up to 99× online and 34× offline runtime speedups. Compared with PRAG, it achieves 19×–58× online and 2×–16× offline end-to-end speedups.

1.1 Related Work

Secure LLM Inference. Secure LLM inference provides the natural execution context for private vector similarity search in RAG. Several works study multi-server secure inference for transformers and LLMs [1, 6, 10, 30]. At the same time, a mainstream approach is to run secure inference with two computing servers, as in Iron [12], BOLT [39], CipherGPT [16], Nimbus [31], BumbleBee [32], and SIGMA [11]. Their linear layers are typically evaluated over two-party additive secret shares. We therefore position CoSecRAG in the same two-party additive secret sharing setting, which makes the vector-similarity-search component easier to integrate into existing secure LLM inference pipelines.

Private RAG and Privacy-Preserving Similarity Retrieval over Vector Databases. Private retrieval for RAG is closely related to privacy-preserving similarity retrieval and approximate nearest-neighbor

retrieval over vector databases. A large body of prior work can be applied directly, or with modest adaptation, to private retrieval for RAG. However, most existing approaches mainly focus on protecting the query side and do not protect the database privacy required in our setting. Examples include RAG-oriented query privacy mechanisms and private retrieval systems [7, 13, 47], as well as privacy-preserving similarity retrieval and private k-nearest-neighbor protocols over vector databases [5, 43, 44, 56]. Another recent line of work studies database privacy in RAG retrieval, for example through differential privacy, random projection, synthetic text generation, or provable privacy guarantees for the released retrieval outputs [28, 35, 50, 53, 55]. These works address an important part of the problem, but they do not simultaneously protect the query privacy in the retrieval protocol. Some approaches can protect both sides, but they are mainly based on homomorphic encryption or encrypted k-nearest-neighbor evaluation [2, 45, 56]. Such techniques are useful in their own settings, but provide limited support for incorporating private vector similarity search into secret-sharing-based LLM workflows.

To our knowledge, PRAG [57] and P^{2}RAG [34] are the only prior secret-sharing-based systems for private vector similarity search in RAG that protect both the database and the query. PRAG is built on Shamir secret sharing and primarily targets an honest-majority setting with more than two servers, where secure vector inner products are inexpensive. Although it can also run in a two-server additive-sharing setting, secure inner products incur substantially higher overhead. In addition, its secure IVF (Inverted File Index) design [20] privately extracts many clusters of secret-shared vectors, which introduces substantial overhead. P^{2}RAG [34] is the most recent two-server baseline for private vector similarity search in RAG. It also targets two-party additive secret sharing, but directly computes the inner-product similarity score between the query and every database vector over secret shares, and then performs top-K selection over all similarity scores. Both steps introduce substantial overhead. Our protocol addresses these bottlenecks and achieves substantially better performance.

2 PRELIMINARIES

2.1 Problem Description

We study private vector similarity search as the first step of relevant-document retrieval in a RAG system. Let $\mathbf{X} = \{\vec{x}_1, \ldots, \vec{x}_N\}$ be a database of document embeddings, where each $\vec{x}_i \in \mathbb{R}^d$ is associated with a document identifier id_i, and let the query user hold a query embedding $\vec{q} \in \mathbb{R}^d$. The vector similarity search goal is to return the top-K identifier set ID_K corresponding to the nearest database vectors to $\vec{q}$ under the chosen similarity or distance measure.

Neural information retrieval systems use several related similarity or distance measures, most commonly dot product, cosine similarity, and Euclidean distance [41]. In this paper, we consider the normalized embedding regime, where all database and query embeddings are ℓ_2-normalized, as commonly done in dense retrieval and privacy-preserving RAG systems [34, 41, 57]:

$$\|\vec{x}_i\|_2 = 1, \qquad \|\vec{q}\|_2 = 1.$$

Under this assumption, all three criteria reduce to the same inner product:

$$\cos(\vec{q}, \vec{x}_i) = \langle \vec{q}, \vec{x}_i \rangle, \qquad \|\vec{q} - \vec{x}_i\|_2^2 = 2 - 2\langle \vec{q}, \vec{x}_i \rangle.$$

Therefore, maximizing dot product, maximizing cosine similarity, and minimizing Euclidean distance are equivalent to selecting the K document embeddings with the largest scores $s_i = \langle \vec{x}_i, \vec{q} \rangle$. Thus, the vector similarity search output is

$$\mathsf{ID}_K = \{\mathsf{id}_i \mid i \in \text{Top-}K_{j \in [N]} \langle \vec{q}, \vec{x}_j \rangle\}.$$

Here Top-K denotes a fixed deterministic operator that returns K distinct indices corresponding to the K largest indexed scores. We define Top-L analogously. The notation $[N]$ denotes the integer set $\{1, \ldots, N\}$; we use the same notation for other positive integers throughout the paper.

In CoSecRAG, we focus on returning document identifiers rather than the document contents themselves. Once the identifiers are obtained, the corresponding documents can be fetched from the document store using PIR techniques [14]. This document-fetching layer is orthogonal to our private vector similarity search protocol, and we do not discuss it further.

CoSecRAG employs a clustering-based vector similarity search algorithm [20]. We write $\mathsf{ID}_K^\star$ for the clustered top-K identifier set returned by CoSecRAG, which is formally defined in Section 3.2.

The output size K is fixed before setup and is treated as a public parameter throughout a protocol instance. All queries processed under the same protocol instance use the same value of K. Supporting a different value of K requires a new protocol instance with the corresponding K-dependent preprocessing material.

2.2 Security Model and Goals

CoSecRAG involves four logical roles: the data owner $\mathcal{D}$, the query user $\mathcal{U}$, and two servers $\mathcal{S}_0, \mathcal{S}_1$. All parties are semi-honest, and the three outsourced-side parties $\mathcal{D}, \mathcal{S}_0, \mathcal{S}_1$ do not collude. The data owner holds the documents, their identifiers, and embeddings, performs offline setup, and stays offline during query processing. The query user issues queries and obtains the final vector similarity search output $\mathsf{ID}_K^\star$. We allow $\mathcal{U}$ to collude with either $\mathcal{S}_0$ or $\mathcal{S}_1$, but not with both servers simultaneously. Such a coalition already knows the query and the prescribed vector similarity search output, but should learn no additional information about the database.

In an end-to-end secure RAG system, private vector similarity search is typically an intermediate module rather than a complete standalone service. The query vector may already be held in additive shares by $\mathcal{S}_0$ and $\mathcal{S}_1$, as the output of an upstream secure computation module executed by the same two servers. Before use by CoSecRAG, these incoming query shares must either be freshly generated for the current protocol invocation or securely rerandomized. Similarly, the retrieved identifier shares may be passed directly to a downstream secure document-fetching module. The resulting document-content shares may then be consumed by a secure LLM inference module also run by $\mathcal{S}_0$ and $\mathcal{S}_1$. Therefore, in a practical deployment, there may not be a real query user that continuously interacts with the CoSecRAG protocol.

We nevertheless introduce a logical query user $\mathcal{U}$ to specify the source of the vector similarity search input and the recipient of its identifier output, and to reflect that the overall RAG request is

ultimately initiated by some user or application. Apart from providing the vector similarity search input and receiving its identifier output, $\mathcal{U}$ does not participate in the internal protocol interaction. This non-interactive query user design makes CoSecRAG easier to compose with end-to-end privacy-preserving RAG pipelines, and is one advantage over P^{2}RAG [34].

The security goal is that, except for the output $\mathsf{ID}_K^\star$ learned by the query user, nothing is revealed beyond the public setup state and query schedule. Intuitively, this public information includes deployment-level parameters such as the database size, embedding dimension, clustering parameters, and fixed output size K. Section 4.1 gives its precise definition and the corresponding security analysis.

2.3 Additive Secret Sharing

The protocol works over the ring $R = \mathbb{Z}_{2^\ell}$. For a value $x \in R$, its two-party additive sharing is written as $[\![x]\!] = (x_0, x_1)$, where $\mathcal{S}_b$ holds x_b and

$$x = x_0 + x_1 \bmod 2^\ell.$$

Each share is uniformly distributed over R. Under the non-collusion assumption, it information-theoretically hides the shared value. We use the same notation componentwise for vectors and matrices.

To secret share a value x means to send the two shares x_0 and x_1 to $\mathcal{S}_0$ and $\mathcal{S}_1$, respectively. To reconstruct x, the servers exchange their shares and add them modulo 2^ℓ.

Additive sharing directly supports local linear operations over $\mathbb{Z}_{2^\ell}$. Given $[\![x]\!]$ and $[\![y]\!]$, the servers can locally compute $[\![x + y]\!]$ and $[\![x - y]\!]$. Given a public scalar c, they can locally compute $[\![cx]\!]$. More generally, linear operations with public values can be carried out locally on shares.

Multiplication of two secret-shared values is not free. We denote the corresponding secure multiplication primitive by Π_{mult}; it can be instantiated using Beaver triples [3]. CoSecRAG is designed to maximize the use of multiplication by public scalars and thereby avoid secret-secret multiplications as much as possible.

2.4 Fixed-Point Encoding and Precision

CoSecRAG represents real-valued embeddings as signed fixed-point integers in $\mathbb{Z}_{2^\ell}$. With f fractional bits, each coordinate is encoded as

$$\hat{x}_{i,j} = \lfloor 2^f x_{i,j} \rceil, \qquad \hat{q}_j = \lfloor 2^f q_j \rceil.$$

The encoded inner product is $s_i = \sum_{j=1}^{d} \hat{x}_{i,j}\hat{q}_j$, whose scale is 2^{2f}. Since all encoded inner-product scores use the same scale and are only used as comparison keys, their order is preserved by integer comparison.

In the secret-shared setting, fixed-point encoding is performed before sharing. The servers hold shares of encoded ring elements, and all arithmetic is interpreted modulo 2^ℓ. Negative encoded integers are represented in $\mathbb{Z}_{2^\ell}$ by two's complement embedding, i.e., an integer z is stored as $z \bmod 2^\ell$. Under this interpretation, ring addition and subtraction agree with signed integer addition and subtraction as long as no modular wraparound occurs. Therefore, correctness is preserved if the score range fits in the signed ℓ-bit representation. A sufficient condition is $d \cdot 2^{2f} < 2^{\ell-1}$, or equivalently, ℓ is chosen large enough, for example

$$\ell \geq 2f + \lceil \log_2 d \rceil + 2.$$

Under this condition, score computation in $\mathbb{Z}_{2^\ell}$ preserves the intended signed fixed-point semantics. In this paper, all plaintext computations and local operations on shares are performed modulo 2^ℓ; with $f = 16$ and $\ell = 64$, this arithmetic can be implemented by native machine-word overflow. This encoding is also used in prior secure LLM inference systems [26, 49].

2.5 Core Primitives

Selection Primitive. We abstract secure top-K selection as a functionality

$$\{[\![\vec{b}^{(r)}]\!]\}_{r\in[K]} \leftarrow \Pi_{\mathsf{top}}([\![\vec{s}]\!], K), \qquad [\![\vec{s}]\!] \in \mathbb{Z}_{2^\ell}^m.$$

Here $[\![\vec{s}]\!]$ is a secret-shared score vector and K is public. Each plaintext selector $\vec{b}^{(r)} = (b_1^{(r)}, \ldots, b_m^{(r)})$ is one-hot. Let i_r denote its unique nonzero position, i.e.,

$$b_i^{(r)} = 1 \iff i = i_r.$$

The selectors satisfy

$$\{i_r \mid r \in [K]\} = \mathsf{Top}\text{-}K_{j\in[m]} s_j.$$

Thus, they identify exactly K distinct selected positions. The functionality keeps the selected positions hidden from each server and is used as a building block in both cluster-level and candidate-level top-K retrieval. In our implementation, this selection primitive is realized using Yao's garbled circuits [52, 54].

Extraction Primitive. Extraction is a primitive for privately selecting one element or vector from a sequence of secret-shared inputs according to a secret-shared one-hot selector.

Let $[\![\vec{b}]\!] = ([\![b_1]\!], \ldots, [\![b_M]\!])$ be a secret-shared one-hot selector, and let i_{ext} be the unique index such that $b_{i_{\mathsf{ext}}} = 1$. Given a sequence of secret-shared elements $[\![\vec{x}]\!] = ([\![x_1]\!], \ldots, [\![x_M]\!])$, single-element extraction is defined as

$$[\![z]\!] \leftarrow \Pi_{\mathsf{ext}}([\![\vec{b}]\!]; [\![\vec{x}]\!]), \qquad [\![z]\!] = [\![x_{i_{\mathsf{ext}}}]\!].$$

Thus, the functionality extracts exactly one element while keeping the selected position hidden from each server.

The same idea extends to vector-valued inputs. For batched extraction, let $[\![\mathbf{X}]\!] = ([\![\vec{x}_1]\!], \ldots, [\![\vec{x}_M]\!])$ contain M aligned input vectors, where $\vec{x}_i = (x_{i,1}, \ldots, x_{i,B})$. The batched functionality is

$$[\![\vec{z}]\!] \leftarrow \Pi_{\mathsf{extB}}([\![\vec{b}]\!]; [\![\mathbf{X}]\!]), \qquad [\![\vec{z}]\!] = [\![\vec{x}_{i_{\mathsf{ext}}}]\!].$$

Extraction can be implemented by multiplying secret-shared one-hot bits with secret-shared ring elements using standard oblivious transfer (OT)-based mixed multiplication techniques [9, 17].

We use Π_{ext} and Π_{extB} in top-K computation to extract the results returned by Π_{top}.

3 COSECRAG PROTOCOL

The workflow of CoSecRAG consists of two main modules. The first computes the query–database scores, while the second performs deterministic clustered top-K selection and returns $\mathsf{ID}_K^\star$ to the query user. We provide efficient constructions for both modules. This section first presents the two protocol components in detail and

then combines them into a complete private vector similarity search protocol.

3.1 IPQ-Mask for Score Computation

In secret-sharing-based private vector similarity search, the main online scoring bottleneck is the secure multiplication needed when both the query vector and the database vectors are secret-shared [34, 57]. A direct secure multiparty computation (MPC) implementation computes each score $s_i = \langle \vec{x}_i, \vec{q} \rangle$ from shares of both inputs, requiring secure multiplications across all document-coordinate pairs. This cost becomes dominant when the database contains millions of vectors.

CoSecRAG addresses this bottleneck by constructing an *Inner Product Preserving Query Mask* (IPQ-Mask). This mask turns online score computation into local linear operations with a public masked-query difference, thereby avoiding online secure multiplications. An IPQ-Mask is a database-dependent and query-independent masking object prepared before the query arrives. Although described for $\mathbf{X}$ below, the construction applies unchanged to any fixed matrix with d columns. For a fixed matrix $\mathbf{X}$, an IPQ-Mask instance consists of a one-time query mask $\vec{a} \in \mathbb{Z}_{2^\ell}^d$ and its matrix response

$$\vec{u} = \mathbf{X}\vec{a}.$$

In plaintext, for any query vector $\vec{q}$, if the servers later open only the masked difference

$$\vec{e} = \vec{q} - \vec{a},$$

then all database inner-product scores can be recovered as

$$\mathbf{X}\vec{q} = \mathbf{X}\vec{a} + \mathbf{X}(\vec{q} - \vec{a}) = \vec{u} + \mathbf{X}\vec{e}.$$

This identity is the source of the term "inner product preserving": the opened vector $\vec{e}$ does not itself preserve inner products, but the correlated pair $(\vec{a}, \vec{u} = \mathbf{X}\vec{a})$ preserves the ability to compute all inner products between $\mathbf{X}$ and $\vec{q}$.

In the secret-shared setting, the servers hold additive shares $[\![\mathbf{X}]\!]$, $[\![\vec{a}]\!]$, $[\![\vec{u}]\!]$, and later receive additive shares $[\![\vec{q}]\!]$. They first compute and open

$$[\![\vec{e}]\!] = [\![\vec{q}]\!] - [\![\vec{a}]\!].$$

The opening of $\vec{e}$ is safe because $\vec{a}$ is sampled uniformly from the full ring $\mathbb{Z}_{2^\ell}^d$. For any fixed $\vec{q}$, the value $\vec{e} = \vec{q} - \vec{a}$ is also uniformly distributed over $\mathbb{Z}_{2^\ell}^d$, and is therefore information-theoretically independent of $\vec{q}$. After $\vec{e}$ is opened, the servers locally compute

$$[\![\vec{s}]\!] = [\![\vec{u}]\!] + [\![\mathbf{X}]\!]\vec{e}.$$

Correctness follows from linearity of additive sharing:

$$[\![\vec{s}]\!] = [\![\vec{u}]\!] + [\![\mathbf{X}]\!]\vec{e} = [\![\mathbf{X}\vec{a}]\!] + [\![\mathbf{X}]\!](\vec{q} - \vec{a}) = [\![\mathbf{X}\vec{q}]\!].$$

Thus, the servers obtain additive shares of the full-score vector $\vec{s} = \mathbf{X}\vec{q}$, while each individual server sees only shares of $\mathbf{X}$, $\vec{a}$, $\vec{u}$, and $\vec{s}$, together with the public one-time masked value $\vec{e}$. Since $\vec{e}$ is public, the multiplication $[\![\mathbf{X}]\!]\vec{e}$ is a secret-shared matrix multiplied by a public vector and requires only local ring operations. No online secure multiplication is needed for full-database score computation.

3.1.1 Offline Preprocessing by the Data Owner. For each future query, the data owner constructs one IPQ-Mask instance. The data owner samples a full-ring mask $\vec{a} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d$, computes the database response $\vec{u} = \mathbf{X}\vec{a}$, and secret shares both $\vec{a}$ and $\vec{u}$ to the two servers.

Each IPQ-Mask instance is consumed at most once. If the same mask $\vec{a}$ were reused for two queries $\vec{q}$ and $\vec{q}'$, then the two opened masked differences $\vec{e} = \vec{q} - \vec{a}$ and $\vec{e}' = \vec{q}' - \vec{a}$ would reveal $\vec{e} - \vec{e}' = \vec{q} - \vec{q}'$. Therefore, the servers maintain a usage state for each mask instance and mark it as consumed after one online execution. Algorithm 1 formalizes this offline phase.

Algorithm 1 mask = $([\![\vec{a}]\!], [\![\vec{u}]\!]) \leftarrow$ IPQMask($\mathbf{X}$)

Input: Fixed matrix $\mathbf{X} \in \mathbb{Z}_{2^\ell}^{r \times d}$ with public row count r.
Output: One-time IPQ-Mask mask = $([\![\vec{a}]\!], [\![\vec{u}]\!])$.
Executed by $\mathcal{D}$.

1: Sample a full-ring random query mask $\vec{a} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d$.
2: Compute the matrix response $\vec{u} \leftarrow \mathbf{X}\vec{a}$.
3: mask = $([\![\vec{a}]\!], [\![\vec{u}]\!])$.

3.1.2 Online Score Computation by the Servers. In the online phase of this subprotocol, the servers already hold shares $[\![\mathbf{X}]\!]$ and an unused IPQ-Mask instance $([\![\vec{a}]\!], [\![\vec{u}]\!])$. They have also received the secret-shared query vector $[\![\vec{q}]\!]$ from the complete protocol. The servers consume the selected IPQ-Mask instance, locally subtract the mask shares from the query shares, and open the masked difference $\vec{e}$. They then use the public vector $\vec{e}$ to compute the full-score vector locally.

Concretely, each server $\mathcal{S}_i$ first computes

$$[\![\vec{e}]\!]_i = [\![\vec{q}]\!]_i - [\![\vec{a}]\!]_i.$$

The servers reconstruct $\vec{e}$ from $[\![\vec{e}]\!]$. Since $\vec{e}$ is public after this step, each server can compute its output score share by a local matrix-vector multiplication:

$$[\![\vec{s}]\!]_i = [\![\vec{u}]\!]_i + [\![\mathbf{X}]\!]_i\vec{e}.$$

No score is opened. The output of this subprotocol is the secret-shared score vector $[\![\vec{s}]\!]$, which is then used by the subsequent secure top-K subprotocol. Algorithm 2 formalizes this online score-computation phase.

Algorithm 2 $[\![\vec{s}]\!] \leftarrow$ Score($[\![\vec{q}]\!], [\![\mathbf{X}]\!]$, mask)

Input: Servers hold $[\![\mathbf{X}]\!]$, unused mask = $([\![\vec{a}]\!], [\![\vec{u}]\!])$, and query shares $[\![\vec{q}]\!]$.
Output: Servers obtain shares $[\![\vec{s}]\!]$ of $\vec{s} = \mathbf{X}\vec{q}$.
Executed by $\mathcal{S}_0$ and $\mathcal{S}_1$.

1: Mark mask as consumed.
2: Locally compute $[\![\vec{e}]\!] \leftarrow [\![\vec{q}]\!] - [\![\vec{a}]\!]$.
3: Reconstruct $\vec{e}$.
4: Locally compute $[\![\vec{s}]\!] \leftarrow [\![\vec{u}]\!] + [\![\mathbf{X}]\!]\vec{e}$.

Table 1: Cost comparison for full-database scoring.

Component	Beaver triples	IPQ-Mask
Offline material	Nd triples	$\vec{a}, \vec{u} = \mathbf{X}\vec{a}$
Offline communication	$6Nd$ shares	$2(N + d)$ shares
Online opening	$2Nd$ elements	d elements
Online communication	$4Nd$ shares	$2d$ shares
Online secure multiplication	Nd	0
Extra offline computation	Nd triples	plaintext $\mathbf{X}\vec{a}$

3.1.3 Benefits of IPQ-Mask. We compare IPQ-Mask with Beaver-triple-based secure multiplication [3] for full-database scoring in the two-party additive-sharing setting [34]. A Beaver multiplication consumes one random triple (a, b, c) with $c = ab$ prepared offline. Online, the servers open two masked differences and use them to reconstruct the product. A direct secure implementation of all document-query inner products therefore requires Nd Beaver multiplications.

Offline communication. Beaver-based scoring requires Nd triples. Each triple contains three ring elements, and distributing shares to two servers costs $6Nd$ share elements. IPQ-Mask sends shares of one mask vector $\vec{a} \in \mathbb{Z}_{2^\ell}^d$ and one response vector $\vec{u} = \mathbf{X}\vec{a} \in \mathbb{Z}_{2^\ell}^N$, for a total of $2(N + d)$ share elements.

Online communication. Each Beaver multiplication opens two masked differences, so full scoring opens $2Nd$ ring elements. Under two-party opening, this corresponds to $4Nd$ share elements exchanged between the servers. IPQ-Mask opens only the masked-query difference of length d, $\vec{e} = \vec{q} - \vec{a}$, which costs d ring elements, or $2d$ share elements.

Computation. IPQ-Mask avoids reconstructing the masked differences and products required by Beaver-based multiplication, reducing the online constant factor. Its cost is one additional offline plaintext matrix-vector product $\vec{u} = \mathbf{X}\vec{a}$ by the data owner. This computation is entirely plaintext and inexpensive compared with the reduction in online MPC communication and secure multiplication.

Table 1 summarizes the concrete cost comparison. In representative retrieval workloads, N ranges from thousands to millions [46], while commonly used embedding dimensions d range from 768 to 3072 [37, 41]. At this scale, removing most of the Nd secure multiplication terms substantially improves the practical performance of score computation. At $N = 2^{22}$, our score-computation experiment reports a 181× online speedup and a 101× offline speedup over P^{2}RAG [34] (Section 5.2).

3.2 Post-Scoring Cluster Pruning for Secure Top-K

IPQ-Mask removes the secure vector-inner-product bottleneck from score computation, making full-database scoring very efficient in terms of online communication and interaction. We now turn to the remaining bottlenecks in private vector similarity search for RAG and address them together. A full-database secure top-K selection approach, as used by P^{2}RAG [34], computes all database scores and then performs secure top-K selection over the full score vector, which introduces a large secure-comparison cost. PRAG [57] avoids this full-database top-K selection by using a cluster-first strategy, but it must privately extract selected clusters containing many high-dimensional document vectors, which also introduces substantial cost.

This subsection addresses both bottlenecks through *Post-Scoring Cluster Pruning* (PSCP). PSCP keeps the low-cost full-database scoring enabled by IPQ-Mask, and then performs cluster pruning after scores have been computed. As a result, secure selection is applied only to a cluster-pruned candidate set, and private extraction operates on scalar score records rather than high-dimensional embeddings.

Before describing PSCP, we first recall the plaintext cluster-pruning strategy. In plaintext retrieval, a useful way to reduce the search space is clustering, as in inverted file and related approximate nearest neighbor indexes [20]. Let the database indices be partitioned into M disjoint clusters:

$$[N] = C_1 \cup \cdots \cup C_M, \qquad C_t \cap C_{t'} = \emptyset \text{ for } t \neq t'.$$

For each cluster C_t, let $\vec{\mu}_t$ be its cluster centroid. The centroid matrix is

$$\mathbf{C} = (\vec{\mu}_1, \ldots, \vec{\mu}_M)^\top.$$

Given a query $\vec{q}$, plaintext cluster-pruned retrieval first selects the top L clusters:

$$\mathcal{T}_L(\vec{q}) = \text{Top-}L_{t\in[M]}\langle \vec{\mu}_t, \vec{q}\rangle.$$

It then searches the records in the selected clusters:

$$C(\vec{q}) = \bigcup_{t\in\mathcal{T}_L(\vec{q})} C_t.$$

The clustered top-K positions and identifier set are

$$\mathcal{R}_K(\vec{q}) = \text{Top-}K_{i\in C(\vec{q})}\langle \vec{x}_i, \vec{q}\rangle, \qquad \mathsf{ID}_K^\star = \{\mathsf{id}_i \mid i \in \mathcal{R}_K(\vec{q})\}.$$

Both Top-L and Top-K use the fixed deterministic operators defined in Section 2. Thus, if each cluster has capacity B, the final top-K selection is performed over at most LB candidate records instead of all N records. In general, $\mathsf{ID}_K^\star$ may differ from ID_K; they are equal whenever all exact global top-K positions occur in $C(\vec{q})$.

CoSecRAG also uses clustering to avoid secure top-K selection over all database scores, as required by P^{2}RAG [34]. However, PSCP changes the prune-then-score order used in plaintext retrieval [20, 22] and in PRAG [57]. In that order, cluster pruning is performed before full-score computation: one first selects several clusters, extracts the vectors inside these clusters, and then computes query-document scores only over the extracted vectors.

The key observation is that the relative costs are reversed in our secure setting. In plaintext retrieval, extracting the vectors inside selected clusters typically incurs substantially lower cost than computing query-document scores over many vectors. After IPQ-Mask, however, full-database score computation requires only one masked opening and local matrix-vector multiplication, while private extraction of encrypted document vectors remains expensive because it must move d-dimensional payloads through secure extraction.

Therefore, PSCP moves cluster pruning after score computation and extracts only scalar score records. It first computes the full document score vector $[\![\vec{s}]\!] = [\![\mathbf{X}\vec{q}]\!]$ using IPQ-Mask, and then uses the cluster structure to derive a smaller subset of scores for secure top-K selection. This avoids the expensive extraction of secret-shared high-dimensional document vectors.

Concretely, PSCP receives secret-shared centroid scores $[\![\vec{cs}]\!] = [\![\mathbf{C}\vec{q}]\!]$. The complete protocol produces these scores jointly with the document scores using one IPQ-Mask invocation. The servers invoke $\Pi_{\mathsf{top}}([\![\vec{cs}]\!], L)$ to obtain secret-shared selectors and use Π_{extB} to extract the corresponding candidate-score blocks. After suppressing dummy records, they invoke $\Pi_{\mathsf{top}}([\![\vec{s}']\!], K)$ over the resulting LB scalar scores. Thus, secure selection scales with LB rather than N.

3.2.1 Offline Preprocessing by the Data Owner. The data owner performs clustering before queries arrive [40]. It partitions the database into M balanced clusters $C_1, \ldots, C_M$ and computes one centroid $\vec{\mu}_t$ for each cluster. The cluster centroids form the matrix

$$\mathbf{C} = (\vec{\mu}_1, \ldots, \vec{\mu}_M)^\top \in \mathbb{Z}_{2^\ell}^{M \times d}.$$

To hide the real cluster sizes and keep the online protocol shape fixed, each cluster is padded to a public capacity B. For every cluster t, the data owner constructs a public padded index vector

$$\vec{C'}_t \in [MB]^B.$$

The entries of $\vec{C'}_t$ point to the padded rows assigned to cluster t. This padded layout is public to the servers and is included in the public setup state defined in Section 4.1. Every cluster has the same public capacity B, while the document vectors, identifiers, valid bits, and scores remain secret-shared; hence the layout does not reveal the real cluster sizes. Each real record slot contains a document vector, a document identifier, and a valid bit $v_i = 1$. Each dummy slot contains a dummy vector, a dummy identifier, and a valid bit $v_i = 0$. The dummy vectors may be arbitrary, because dummy records are suppressed before the final top-K selection.

The padded database matrix is denoted by

$$\widetilde{\mathbf{X}} \in \mathbb{Z}_{2^\ell}^{MB \times d}.$$

Scores are computed over this padded layout, and each cluster is represented online by a vector of length B of scalar records aligned with $\vec{C'}_t$. The data owner secret shares $\widetilde{\mathbf{X}}$, $\mathbf{C}$, all document identifiers, and all valid bits to the two servers. The joint per-query IPQ-Mask material is prepared later in the complete protocol. Algorithm 3 formalizes this cluster setup phase.

3.2.2 Online PSCP Top-K by the Servers. In the online phase, the servers already hold document score shares $[\![\vec{s}]\!]$ over the padded layout and centroid score shares $[\![\vec{cs}]\!] = [\![\mathbf{C}\vec{q}]\!]$ from the joint scoring step. The servers then invoke $\Pi_{\mathsf{top}}([\![\vec{cs}]\!], L)$ over the M cluster scores. This selection does not reveal the selected cluster identifiers. Instead, it outputs L secret-shared one-hot selectors $[\![\vec{b}^{(1)}]\!], \ldots, [\![\vec{b}^{(L)}]\!]$ whose nonzero positions collectively identify the selected clusters. These positions are distinct.

Candidate generation uses the batched extraction primitive Π_{extB}. Each online block Blk_t is the vector of length B of scalar records aligned with $\vec{C'}_t$:

$$[\![\mathsf{Blk}_t]\!] = \big([\![R_{t,1}]\!], \ldots, [\![R_{t,B}]\!]\big).$$

Each record consists of a score share, an identifier share, and a valid-bit share:

$$[\![R_{t,j}]\!] = ([\![s_{t,j}]\!], [\![\mathsf{id}_{t,j}]\!], [\![v_{t,j}]\!]).$$

Algorithm 3 $([\![\mathbf{C}]\!], [\![\widetilde{\mathbf{X}}]\!], [\![\mathsf{ID}]\!], [\![\vec{v}]\!], \{\vec{C'}_t\}_{t\in[M]}) \leftarrow \mathsf{ClusterSetup}(\mathbf{X}, \mathsf{ID})$

Input: Database matrix $\mathbf{X} \in \mathbb{Z}_{2^\ell}^{N \times d}$, document identifiers ID, public parameters M and B.

Output: Secret-shared cluster centroids, padded database, identifiers, valid bits, and public padded index vectors.

Executed by $\mathcal{D}$.

1: Partition the database into M balanced clusters $C_1, \ldots, C_M$.
2: **for** $t = 1$ to M **do**
3: Compute the cluster centroid $\vec{\mu}_t$ of cluster C_t.
4: Create a padded index vector $\vec{C'}_t$ of length B.
5: Insert real records $(\vec{x}_i, \mathsf{id}_i, v_i = 1)$ for $i \in C_t$.
6: Pad the remaining slots with dummy records $(\vec{x}_\perp, \mathsf{id}_\perp, v_\perp = 0)$.
7: **end for**
8: Form the padded database matrix $\widetilde{\mathbf{X}}$ and the centroid matrix $\mathbf{C}$.
9: Secret shares $\widetilde{\mathbf{X}}$, $\mathbf{C}$, all identifiers, and all valid bits to $\mathcal{S}_0$ and $\mathcal{S}_1$.

For every selector $[\![\vec{b}^{(r)}]\!]$, the servers extract one secret-shared candidate block:

$$[\![\mathsf{CandBlk}_r]\!] \leftarrow \Pi_{\mathsf{extB}}\big([\![\vec{b}^{(r)}]\!]; \{[\![\mathsf{Blk}_t]\!]\}_{t\in[M]}\big), \quad r \in [L].$$

The cluster index encoded by each selector remains hidden, while the output shape is fixed. The servers concatenate the extracted blocks into a candidate vector:

$$[\![\mathsf{Cand}]\!] = [\![\mathsf{CandBlk}_1]\!] \,\|\, \cdots \,\|\, [\![\mathsf{CandBlk}_L]\!].$$

The candidate vector has fixed size LB. Since every cluster is padded to the same public capacity B, this size reveals no query-dependent information beyond the public parameters L and B. From $[\![\mathsf{Cand}]\!]$, the servers derive an aligned candidate identifier vector $[\![\mathsf{CandID}]\!] = ([\![\mathsf{id}'_1]\!], \ldots, [\![\mathsf{id}'_{LB}]\!])$. After dummy records are suppressed, the corresponding score vector is denoted by $[\![\vec{s}']\!]$.

Algorithm 4 formalizes this online cluster top-K phase. Lines 6–10 perform dummy suppression. The MIN-based step is backend-agnostic; for example, one may set $\mathsf{MIN} = -2^{\ell-1}$, which is smaller than every valid encoded score. Our Yao implementation instead incorporates validity bits into secure selection.

3.2.3 Benefits of PSCP. The main benefit of PSCP is that it reduces the two remaining costs in private vector similarity search for RAG: PRAG's secure extraction of high-dimensional vectors [57] and P^{2}RAG's secure selection over all database scores [34].

We first consider the selection cost. Let $C_{\mathsf{sel}}(N, K)$ denote the cost of invoking Π_{top} to select the top K elements from N secret-shared scores. A direct protocol runs one selection over the full database: $C_{\mathsf{direct}} = C_{\mathsf{sel}}(N, K)$. PSCP first selects L clusters from M cluster scores and then selects the final K records from LB candidates: $C_{\mathsf{cluster}} = C_{\mathsf{sel}}(M, L) + C_{\mathsf{sel}}(LB, K)$. In a balanced setting, we choose M and B as powers of two close to $\sqrt{N}$ and use a small constant L, e.g., $L = 8$:

$$M = B = 2^{\lceil \log_2 \sqrt{N} \rceil}, \qquad L = 8.$$

Then the final secure top-K input size becomes $LB \approx 8\sqrt{N}$, instead of N. Since secure selection cost is typically close to linear in the

Algorithm 4 $[\![\mathsf{ID}_K^\star]\!] \leftarrow \mathsf{ClusterTopK}\big([\![\vec{cs}]\!], \{[\![\mathsf{Blk}_t]\!]\}_{t\in[M]}\big)$

Input: Centroid score shares $[\![\vec{cs}]\!]$, padded online blocks $[\![\mathsf{Blk}_1]\!], \ldots, [\![\mathsf{Blk}_M]\!]$, public parameters M, B, L, K, and a public sentinel MIN.

Output: Secret shares of a set of K document identifiers.

Executed by $\mathcal{S}_0$ and $\mathcal{S}_1$.

$\{[\![\vec{b}^{(r)}]\!]\}_{r\in[L]} \leftarrow \Pi_{\mathsf{top}}([\![\vec{cs}]\!], L)$.
for $r = 1$ to L **do**
  $[\![\mathsf{CandBlk}_r]\!] \leftarrow \Pi_{\mathsf{extB}}\big([\![\vec{b}^{(r)}]\!]; \{[\![\mathsf{Blk}_t]\!]\}_{t\in[M]}\big)$.
end for
Form $[\![\mathsf{Cand}]\!]$ and the aligned identifier vector $[\![\mathsf{CandID}]\!]$ from $[\![\mathsf{CandBlk}_1]\!], \ldots, [\![\mathsf{CandBlk}_L]\!]$.
for $r = 1$ to L **do**
  for $b = 1$ to B **do**
    $[\![s'_{r,b}]\!] \leftarrow \mathsf{MIN} + [\![v_{r,b}]\!] \cdot ([\![s_{r,b}]\!] - \mathsf{MIN})$.
  end for
end for
Let $[\![\vec{s'}]\!]$ be the vector of length LB containing all $[\![s'_{r,b}]\!]$ values in the same order as $[\![\mathsf{CandID}]\!]$.
$\{[\![\vec{c}^{(r)}]\!]\}_{r\in[K]} \leftarrow \Pi_{\mathsf{top}}([\![\vec{s'}]\!], K)$.
$[\![\mathsf{id}_{i_r}]\!] \leftarrow \Pi_{\mathsf{ext}}([\![\vec{c}^{(r)}]\!]; [\![\mathsf{CandID}]\!])$ for $r \in [K]$.
return $\{[\![\mathsf{id}_{i_r}]\!]\}_{r\in[K]}$.

input size for a fixed K, this reduces the dominant selection term by roughly

$$\frac{LB}{N} \approx \frac{8\sqrt{N}}{N} = \frac{8}{\sqrt{N}}.$$

The cost is one extra round of cluster-level extraction compared with unclustered retrieval like P^{2}RAG [34]: PSCP extracts L blocks of records containing three scalar fields before secure selection. This extraction is much cheaper than secure selection. Table 2 summarizes the selection and extraction costs of unclustered and post-scoring pruning retrieval. In our top-K experiment, PSCP achieves online speedups of up to 67× over our unclustered variant and up to 81× over P^{2}RAG. Including offline costs, the corresponding total-runtime speedups reach 102× and 49× (Section 5.3).

Table 2: Unclustered vs. post-scoring pruning.

	Unclustered	PSCP
Selection	$C_{\mathsf{sel}}(N, K)$	$C_{\mathsf{sel}}(M, L) + C_{\mathsf{sel}}(LB, K)$
Extraction	$C_{\mathsf{ext}}(N, K)$	$3C_{\mathsf{ext}}(MB, L) + C_{\mathsf{ext}}(LB, K)$

From the extraction perspective, the contrast with PRAG's prune-before-score retrieval [57] is the payload size. This retrieval order selects clusters before document scoring and therefore must privately extract the document embeddings inside the selected clusters. The extracted payload is d-dimensional per document. PSCP reverses this order: after IPQ-Mask produces the full score vector, cluster pruning extracts only scalar score records. Each extracted record contains only a score share, an identifier share, and a valid-bit share, rather than a d-dimensional embedding. Thus, for the first extraction round, PSCP reduces the extracted payload to

$$\frac{3}{d}$$

of the pre-score pruning payload. The cost is that PSCP computes scores for the full database, but after IPQ-Mask this score computation is substantially less costly than privately extracting high-dimensional document vectors because it consists mostly of local arithmetic. Table 3 contrasts PSCP with PRAG's prune-before-score order [57]. For compactness, the table omits the common cost of scoring M centroids and uses $MB \approx N$; let $C_e = C_{\mathsf{ext}}$. This yields 19× to 58× online and 2× to 16× offline end-to-end speedups over PRAG in our experiments (Section 5.4).

Table 3: Prune-before-score vs. post-scoring pruning.

	Pre-score	PSCP
Π_{ext}	$dC_e(MB, L) + C_e(LB, K)$	$3C_e(MB, L) + C_e(LB, K)$
Scoring	LB	N

In terms of retrieval quality, clustering is a widely used approximation technique for large-scale retrieval [22]; we evaluate its retrieval quality on real datasets in Section 5.1.

3.3 Complete Protocol

The complete protocol composes the two components above. The offline phase prepares the padded clustered database and one IPQ-Mask instance per query. The online phase jointly computes document and centroid scores and returns the identifier set $\mathsf{ID}_K^\star$.

In the offline phase, the data owner fixed-point encodes the database embeddings and runs $\mathsf{ClusterSetup}(\mathbf{X}, \mathsf{ID})$. This produces the padded database $\widetilde{\mathbf{X}}$, the centroid matrix $\mathbf{C}$, and the public padded index vectors $\{\vec{C'}_t\}_{t\in[M]}$. The data owner secret shares $\widetilde{\mathbf{X}}$, $\mathbf{C}$, document identifiers, and valid bits to the two servers.

For joint scoring, define the row-wise concatenation

$$\mathbf{Z} = \begin{bmatrix} \widetilde{\mathbf{X}} \\ \mathbf{C} \end{bmatrix} \in \mathbb{Z}_{2^\ell}^{(MB+M)\times d}.$$

For each future query, the data owner prepares one IPQ-Mask $\mathsf{mask}_{\mathbf{Z}} = ([\![\vec{a}]\!], [\![\mathbf{Z}\vec{a}]\!])$. The servers form $[\![\mathbf{Z}]\!]$ by concatenating their existing matrix shares.

In the online phase, the user secret shares the encoded query $\vec{q}$ to the servers. The servers invoke the score procedure once:

$$[\![\mathbf{t}]\!] \leftarrow \mathsf{Score}\big([\![\vec{q}]\!], [\![\mathbf{Z}]\!], \mathsf{mask}_{\mathbf{Z}}\big).$$

This opens one masked difference and yields $[\![\mathbf{t}]\!] = [\![\vec{s}]\!] \,\|\, [\![\vec{cs}]\!]$, where $[\![\vec{s}]\!] = [\![\widetilde{\mathbf{X}}\vec{q}]\!]$ and $[\![\vec{cs}]\!] = [\![\mathbf{C}\vec{q}]\!]$. They assemble the padded online blocks $\{[\![\mathsf{Blk}_t]\!]\}_{t\in[M]}$ from aligned score, identifier, and valid-bit shares. They then invoke $\mathsf{ClusterTopK}$ with $[\![\vec{cs}]\!]$ to extract candidate blocks and produce $\mathsf{ID}_K^\star$. The servers send only shares of these identifiers to the user, who reconstructs $\mathsf{ID}_K^\star$. Algorithm 5 formalizes the complete protocol.

In an end-to-end secure RAG deployment, this user interface can be replaced by adjacent secure modules: the query vector may arrive already secret-shared from an upstream module. Such incoming query shares must be fresh for the current protocol invocation or

securely rerandomized before use. The retrieved identifier shares may be passed directly to a downstream secure document-fetching module, and the resulting document-content shares may then be consumed by secure LLM inference, as discussed in Section 2.2.

Algorithm 5 $\mathsf{ID}_K^\star \leftarrow \mathsf{CoSecRAG}(\mathbf{X}, \mathsf{ID}, \vec{q})$

Input: Data owner's database $\mathbf{X}$, document identifiers ID, query vector $\vec{q}$, and public parameters $N, d, M, B, L, K, \ell, f$.

Output: The user obtains a set of K document identifiers.

Offline phase.

1: $\mathcal{D}$ fixed-point encodes and normalizes the database embeddings.
2: $\mathcal{D}$ runs $(\llbracket \mathbf{C} \rrbracket, \llbracket \widetilde{\mathbf{X}} \rrbracket, \llbracket \mathsf{ID} \rrbracket, \llbracket \vec{v} \rrbracket, \{\vec{C'}_t\}_{t\in[M]}) \leftarrow \mathsf{ClusterSetup}(\mathbf{X}, \mathsf{ID})$.
3: $\mathcal{D}$ forms $\mathbf{Z} \leftarrow (\widetilde{\mathbf{X}}^\top, \mathbf{C}^\top)^\top$.
4: $\mathcal{D}$ runs $\mathsf{mask}_\mathbf{Z} \leftarrow \mathsf{IPQMask}(\mathbf{Z})$.
5: $\mathcal{D}$ sends $\mathsf{mask}_\mathbf{Z}$ to $\mathcal{S}_0, \mathcal{S}_1$, which mark it as unused.
6: $\mathcal{S}_0, \mathcal{S}_1$ form $\llbracket \mathbf{Z} \rrbracket$ by row-wise concatenating $\llbracket \widetilde{\mathbf{X}} \rrbracket$ and $\llbracket \mathbf{C} \rrbracket$.
7: $\mathcal{S}_0, \mathcal{S}_1$ prepare the Yao and OT-based materials required by ClusterTopK for the fixed public parameters L and K.
8: $\mathcal{D}$ goes offline before the online phase.

Online phase.

9: $\mathcal{U}$ fixed-point encodes and freshly secret shares $\vec{q}$ to $\mathcal{S}_0$ and $\mathcal{S}_1$.
10: $\mathcal{S}_0, \mathcal{S}_1$ compute $\llbracket \mathbf{t} \rrbracket \leftarrow \mathsf{Score}(\llbracket \vec{q} \rrbracket, \llbracket \mathbf{Z} \rrbracket, \mathsf{mask}_\mathbf{Z})$.
11: Split $\llbracket \mathbf{t} \rrbracket = \llbracket \vec{s} \rrbracket \,\|\, \llbracket \vec{cs} \rrbracket$, where $\llbracket \vec{s} \rrbracket$ has length MB and $\llbracket \vec{cs} \rrbracket$ has length M.
12: $\mathcal{S}_0, \mathcal{S}_1$ assemble $\{\llbracket \mathsf{Blk}_t \rrbracket\}_{t\in[M]}$ from $\llbracket \vec{s} \rrbracket$, $\llbracket \mathsf{ID} \rrbracket$, and $\llbracket \vec{v} \rrbracket$.
13: $\mathcal{S}_0, \mathcal{S}_1$ compute $\llbracket \mathsf{ID}_K^\star \rrbracket \leftarrow \mathsf{ClusterTopK}(\llbracket \vec{cs} \rrbracket, \{\llbracket \mathsf{Blk}_t \rrbracket\}_{t\in[M]})$.
14: $\mathcal{S}_0, \mathcal{S}_1$ send $\llbracket \mathsf{ID}_K^\star \rrbracket$ to $\mathcal{U}$.
15: $\mathcal{U}$ reconstructs $\mathsf{ID}_K^\star$.
16: **return** $\mathsf{ID}_K^\star$ to the user.

4 ANALYSIS

Section 3 already analyzes the security and performance of the main subprotocols. This section connects these pieces and gives an overall discussion of the security and efficiency of CoSecRAG.

4.1 Security

We consider static semi-honest adversaries. The two servers do not collude, and the data owner does not collude with either server. We also allow the query user to collude with one server. Such a coalition already knows the query and the prescribed retrieval output, but should learn no additional information about the database. The restriction on collusion between the data owner and a server is necessary in our current construction: the data owner knows the mask $\vec{a}$, while a server observes the opened value $\vec{e} = \vec{q} - \vec{a}$.

Let

$$\mathsf{pp} = \big(N, d, M, B, L, K, \ell, f, \mathsf{MIN}, \mathsf{layout}\big)$$

denote the public setup state. Unlike the remaining components of pp, layout is not fixed before setup: it is generated by ClusterSetup and becomes public when setup completes. It contains the public padded index vectors $\{\vec{C'}_t\}_{t\in[M]}$. The number and timing of query executions are also public. We denote this public query schedule by $\mathcal{L}_{\mathsf{query}}$. For a single-query execution, $\mathcal{L}_{\mathsf{query}} = (\mathsf{invocation})$; for a multi-query execution, it is the public sequence of invocation events, including their number and timing. These values determine the dimensions and communication pattern of the protocol.

Intuitive privacy analysis. From the view of the data owner, no online query-dependent information is observed. The data owner completes the clustered database setup and prepares the IPQ-Mask material before any query arrives. It receives no online messages and therefore learns neither the query nor the selected clusters, scores, candidates, or output identifiers.

From the view of the query user, the only database-dependent output is $\mathsf{ID}_K^\star$. The user does not receive centroid scores, document scores, selected clusters, candidate records, or server-side intermediate values. Information inferred from one or more legitimate outputs is inherent in the vector similarity search functionality rather than additional protocol leakage.

From the view of either individual server, the database embeddings, cluster centroids, identifiers, valid bits, query, scores, selectors, and outputs are additively secret-shared. During IPQ-Mask scoring, the only opened query-dependent value is $\vec{e} = \vec{q} - \vec{a}$. Since $\vec{a}$ is sampled uniformly over the full ring and is used for only one query, $\vec{e}$ is uniformly distributed and statistically independent of $\vec{q}$. All document and centroid scores remain secret-shared. The secure top-K procedures output secret-shared selectors, while the extraction procedures use these selectors without revealing their selected positions. Moreover, every cluster is padded to the same capacity B, so each query processes exactly LB candidate records. Hence the transcript reveals neither the selected clusters nor a query-dependent access or volume pattern.

The same database-privacy argument holds when the query user colludes with one server. The coalition already knows $\vec{q}$ and $\mathsf{ID}_K^\star$, and can derive $\vec{a} = \vec{q} - \vec{e}$. Nevertheless, it still holds only one additive share of each database-dependent value. These shares remain information-theoretically hiding, and the remaining transcripts are protected by the underlying secure computation primitives.

Claim 1. *In the semi-honest model, assuming that the two servers do not collude and that the data owner does not collude with either server, CoSecRAG reveals no information beyond the public parameters, the public query schedule, and the output $\mathsf{ID}_K^\star$ returned to the query user. In particular, no individual server learns the database, query, scores, selected clusters, candidate records, or final identifiers. Even if the query user colludes with one server, the coalition learns no additional database information beyond the prescribed vector similarity search output.*

We next formalize this claim using a plaintext clustered vector similarity search functionality.

Definition 4.1 (Clustered top-K functionality). Let

$$\mathcal{I} = \big(\mathbf{X}, \mathsf{ID}, \{(\vec{\mu}_t, C_t)\}_{t\in[M]}\big)$$

be a fixed clustered index, where $\mathbf{X} = (\vec{x}_1, \ldots, \vec{x}_N)$ is the encoded database, $\mathsf{ID} = (\mathsf{id}_1, \ldots, \mathsf{id}_N)$, $\vec{\mu}_t$ is the centroid of cluster t, and the sets $C_1, \ldots, C_M$ partition $[N]$.

The ideal functionality $\mathcal{F}_{\mathsf{ClusteredTopK}}$ is parameterized by pp, where K and L are fixed during setup. It stores $\mathcal{I}$ during setup. Upon receiving an encoded query $\vec{q}$ from $\mathcal{U}$, it assumes that the

selected clusters contain at least K records and computes

$$\mathcal{T}_L(\vec{q}) = \text{Top-}L_{t\in[M]}\langle\vec{\mu}_t, \vec{q}\rangle,$$

forms the candidate set

$$C(\vec{q}) = \bigcup_{t\in\mathcal{T}_L(\vec{q})} C_t,$$

and computes

$$\mathcal{R}_K(\vec{q}) = \text{Top-}K_{i\in C(\vec{q})}\langle\vec{x}_i, \vec{q}\rangle.$$

It sends the identifier set

$$\mathsf{ID}_K^\star = \{\mathsf{id}_i \mid i \in \mathcal{R}_K(\vec{q})\}$$

to $\mathcal{U}$ and sends no private output to $\mathcal{D}, \mathcal{S}_0$, or $\mathcal{S}_1$. Both Top-L and Top-K use the fixed deterministic operators defined in Section 2.

The functionality captures deterministic plaintext cluster-pruned vector similarity search. Its output $\mathsf{ID}_K^\star$ may differ from the exact reference ID_K; this approximation is evaluated separately in Section 5.1. CoSecRAG is required to reproduce the output of $\mathcal{F}_{\mathsf{ClusteredTopK}}$ without adding approximation or protocol leakage.

Theorem 4.2 (Security of CoSecRAG). *Assume that each query is freshly additively shared, or that existing query shares are securely rerandomized before use. Assume further that the masks used for distinct query invocations are sampled independently as*

$$\vec{a} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d$$

and that each mask is used for at most one query. Finally, assume that Π_{top}, Π_{ext}, Π_{extB}, *and* Π_{mult} *securely realize their respective functionalities against semi-honest adversaries.*

Then CoSecRAG securely realizes $\mathcal{F}_{\mathsf{ClusteredTopK}}$ *against each of the following static corruption sets:*

$$\{\mathcal{D}\}, \quad \{\mathcal{U}\}, \quad \{\mathcal{S}_0\}, \quad \{\mathcal{S}_1\}, \quad \{\mathcal{U}, \mathcal{S}_0\}, \quad \{\mathcal{U}, \mathcal{S}_1\}.$$

More precisely, for every allowed corruption set $\mathcal{T}$*, there exists a probabilistic polynomial-time simulator* $\mathsf{Sim}_\mathcal{T}$ *such that*

$$\mathsf{View}_\mathcal{T}^{\Pi_{CoSecRAG}}(\mathcal{I}, \vec{q}; \mathcal{L}_{\mathsf{query}}) \approx_c \mathsf{Sim}_\mathcal{T}\left(1^\lambda, \mathsf{pp}, \mathcal{L}_{\mathsf{query}}, \mathsf{in}_\mathcal{T}, \mathsf{out}_\mathcal{T}\right),$$

where the inputs and outputs are those prescribed by $\mathcal{F}_{\mathsf{ClusteredTopK}}$.

Proof sketch. The data owner's view is simulated by honestly generating the offline setup from its own input, since it receives no online messages. The query user's input and returned identifiers are freshly additively shared, so its view can be simulated from its query and ideal output. For either server, each individual additive share is uniformly random, while the opened value $\vec{e} = \vec{q} - \vec{a}$ is uniform and independent of $\vec{q}$. The remaining selection, extraction, and multiplication transcripts are simulated using the assumed security of their underlying protocols. For a coalition of the query user and one server, the simulator additionally receives the query and ideal output; knowing $\vec{a} = \vec{q} - \vec{e}$ does not reveal the missing database shares. Fixed padding allows the simulator to reproduce all message lengths and protocol calls from public parameters. The theorem follows by the standard sequential composition theorem [4]. The full proof is provided in Appendix A.

4.2 Performance

Let N be the number of database records and d the embedding dimension. The padded clustered database contains MB records, where typically $MB \approx N$ and $N \gg K$. Our analysis reports per-query cost. We exclude one-time setup costs such as database fixed-point encoding and normalization, balanced clustering, database and metadata sharing, and public-key initialization, since they are executed only during system setup and are reused by subsequent queries. We describe the cost by party.

Data owner. The data owner incurs costs only in the offline phase. Its offline communication to the servers is dominated by sending shares of the joint response $\mathbf{Z}\vec{a} = (\widetilde{\mathbf{X}}\vec{a}) \parallel (\mathbf{C}\vec{a})$, which contains $MB + M$ elements and costs $O(MB)$ share elements per prepared query. The dominant computation is $O((MB + M)d) = O(MBd)$ for computing $\mathbf{Z}\vec{a}$, using only plaintext ring operations.

Query user. The query user only participates online. It secret shares a d-dimensional encoded query vector and later reconstructs K returned identifiers. Thus, both its communication and computation are $O(d + K)$. As discussed in Section 2, the query user is only a logical role. When CoSecRAG is used as an intermediate module in an end-to-end secure RAG pipeline run by the same two servers, these input and output transfers incur no extra cost.

Servers. The servers store the offline shares and consume one IPQ-Mask instance per query. For score computation, the only interaction on secret-shared values is opening the d-dimensional masked difference $\vec{e} = \vec{q} - \vec{a}$; all remaining score computation is local. Thus, scoring has communication complexity $O(d)$ and computation complexity $O((MB+M)d) = O(MBd)$ ring operations. The main remaining cost comes from the cluster-based top-K stage:

$$C_{\mathsf{sel}}(M, L) + C_{\mathsf{sel}}(LB, K) + 3C_{\mathsf{ext}}(MB, L) + C_{\mathsf{ext}}(LB, K).$$

In the generic formulation, dummy suppression uses LB secure multiplications, contributing $O(\sqrt{N})$ under our parameterization and omitted above. Our Yao implementation instead incorporates validity into secure selection, with no separate multiplication term. Here C_{sel} denotes the cost of invoking Π_{top}, and C_{ext} denotes the cost of private extraction. Under our implementation, the two selection calls cost

$$O(M + L\log M + LM) + O(LB + K\log(LB) + KLB),$$

and the two extraction calls cost $O(LMB)+O(KLB)$. This implementation makes the selection cost grow with K. Our focus, however, is on reducing the input size to secure selection rather than optimizing the selection primitive itself; for larger K, one can instead use selection implementations whose cost is less sensitive to K.

When $M = B = \sqrt{N}$ and K, L are constants, the selection cost becomes

$$O(\sqrt{N}) + O(\sqrt{N} + \log N) = O(\sqrt{N}),$$

and the extraction cost becomes $O(N)+O(\sqrt{N}) = O(N)$. Although extraction is linear in N, it only performs selector-based routing, while selection performs secure comparisons and has a much larger constant factor.

In the generic formulation, opening $\vec{e} = \vec{q} - \vec{a}$ and a one-round realization of dummy suppression contribute two fixed rounds. In our Yao/OT implementation, validity is handled within secure selection and adds no separate round. The complete implementation

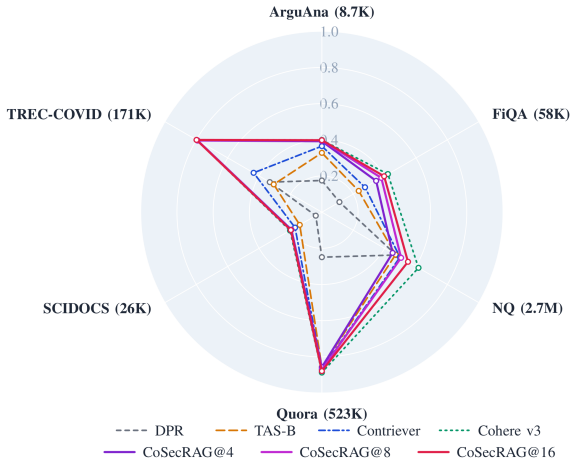


Figure 2: nDCG@10 on six BEIR datasets.

takes 9 communication rounds when query input and result output are excluded, or 11 rounds when they are included.

5 EVALUATION

This section evaluates CoSecRAG through a series of experiments. We first evaluate the retrieval quality of CoSecRAG. This experiment serves two purposes: it shows that the protocol maintains strong retrieval quality, and it validates the chosen clustering configuration before it is used in the subsequent cost experiments. We then measure the costs of score computation and top-K computation separately, highlighting the benefits of IPQ-Mask and PSCP. Finally, we return to the complete CoSecRAG protocol and report its end-to-end vector similarity search performance. Our experiments are based on an C++ implementation of our system. We use libOTe's IKNP implementation for OT extension [17, 42]. The implementation is available in the following public repository [1].

All experiments are run on a single machine with 16 physical Intel(R) Xeon(R) Gold 5122 CPU cores at 3.60 GHz and 512 GB of DDR4 memory. The machine runs Ubuntu 22.04.5 LTS. We instantiate each protocol party as an independent process on the same host. To model a wide-area deployment, we model the communication link between each pair of parties as a 1 Gbps full-duplex link with 20 ms round-trip latency. The scalability experiments use synthetic inputs; for fixed public parameters, their values do not affect the data-oblivious workload.

5.1 Retrieval Quality

Following the BEIR evaluation protocol widely adopted in dense retrieval and RAG-related studies [24, 46], we evaluate retrieval quality measured by nDCG@10 on six BEIR datasets. Ordered by corpus size, these datasets are NQ (2.7M documents), Quora (523K), TREC-COVID (171K), FiQA (58K), SCIDOCS (26K), and ArguAna (8.7K). We compare against the native retrieval performance of four common embedding baselines: DPR [25], TAS-B [15], Contriever [18], and Cohere Embed v3 [8]. We report nDCG@10 (normalized discounted cumulative gain at 10), a standard metric for retrieval evaluation, which measures the quality of the top-10 ranking by rewarding relevant documents placed closer to the top and normalizing the score across queries [19, 46].

For CoSecRAG, we build a 1024-dimensional vector database using Cohere Embed v3 embeddings. For each dataset with N documents, we set $M = B = 2^{\lceil \log_2 \sqrt{N} \rceil}$ and evaluate three cluster-pruning settings, $L \in \{4, 8, 16\}$. Figure 2 compares these CoSecRAG configurations with the baseline performance.

The results show that CoSecRAG maintains strong retrieval quality under the proposed clustering parameters. On most datasets, its nDCG@10 is close to the native Cohere Embed v3 baseline and exceeds the other three classical embedding baselines. On TREC-COVID, the scores of CoSecRAG with $L = 4, 8, 16$ are all above 0.99× the Cohere Embed v3 score. Retrieval quality also generally improves as L increases, but the gain from $L = 4$ to $L = 8$ is larger than the gain from $L = 8$ to $L = 16$, indicating diminishing returns. Thus, in the remaining experiments we set $M = B = 2^{\lceil \log_2 \sqrt{N} \rceil}$ and $L = 8$, with $K = 10$ and $d = 1024$.

Table 4: Score computation runtime as N scales.

N	Online runtime (s)			Offline runtime (s)		
	CoSecRAG	P^{2}RAG	Speedup	CoSecRAG	P^{2}RAG	Speedup
2^{14}	0.07	3.44	49×	0.06	3.42	56×
2^{16}	0.13	13.28	104×	0.16	13.61	85×
2^{18}	0.42	56.03	134×	0.55	54.34	98×
2^{20}	1.46	235.47	162×	2.16	217.36	100×
2^{22}	5.58	1007.08	181×	8.65	869.48	101×

5.2 Score Computation

We next evaluate score computation as the database size N grows from 2^{14} to 2^{22}, comparing CoSecRAG with P^{2}RAG [34]. Table 4 reports both online runtime and per-query offline runtime. The runtime of both protocols increases roughly linearly with N. CoSecRAG is consistently faster: its online speedup ranges from 49× to 181×, and its offline runtime speedup grows from 56× to 101×. The improvement comes from IPQ-Mask, which avoids the communication-heavy inner products between secret-shared vectors. CoSecRAG does not send a large number of Beaver triples in the offline phase, and its online server-to-server communication only opens one masked-query vector: at database size $N = 2^{22}$, the offline communication is reduced from 206.29 GB to 67.16 MB (by 3.07×10^3), while the online communication is reduced from 137.51 GB to 16.38 KB (by 8.39×10^6). These results show the large benefit of IPQ-Mask in reducing secure multiplications, and the advantage will become even more pronounced under poorer network conditions.

5.3 Top-K Computation

We next isolate the cost of secure top-K computation. We compare three variants: CoSecRAG with PSCP, CoSecRAG without clustering, and the top-K procedure of P^{2}RAG [34]. For the PSCP variant,

[1] https://github.com/MyXie2333/CoSecRAG_2026

Table 5: Top-K runtime as N scales.

N	Phase	Runtime (s)			Speedup	
		CoSecRAG	Unclustered	P^{2}RAG	Unclustered	P^{2}RAG
2^{14}	Online	0.62	7.03	10.59	11×	17×
	Total	1.69	23.67	10.61	14×	6×
2^{16}	Online	1.25	26.89	35.70	22×	29×
	Total	3.34	92.28	35.72	28×	11×
2^{18}	Online	3.27	107.79	136.50	33×	42×
	Total	8.06	385.96	136.53	48×	17×
2^{20}	Online	8.48	564.56	534.99	67×	63×
	Total	17.00	1739.86	535.02	102×	31×
2^{22}	Online	26.41	OOM	2145.25	–	81×
	Total	43.58	OOM	2145.28	–	49×

OOM denotes out of memory.

Table 6: End-to-end performance as N scales.

N	Metric	Online			Offline		
		Ours	P^{2}RAG	Ratio	Ours	P^{2}RAG	Ratio
2^{14}	Runtime (s)	0.69	14.02	20×	1.13	3.44	3×
	Comm. (MB)	14.80	546.61	37×	36.52	805.84	22×
2^{16}	Runtime (s)	1.37	48.97	36×	2.25	13.63	6×
	Comm. (MB)	54.73	2,185.53	40×	74.69	3,223.33	43×
2^{18}	Runtime (s)	3.69	192.53	52×	5.34	54.37	10×
	Comm. (MB)	210.08	8,741.23	42×	153.66	12,893.30	84×
2^{20}	Runtime (s)	9.94	770.46	78×	10.68	217.38	20×
	Comm. (MB)	822.79	34,964.02	42×	320.06	51,573.17	161×
2^{22}	Runtime (s)	31.99	3,152.33	99×	25.82	869.50	34×
	Comm. (MB)	3,256.16	139,855.17	43×	682.36	206,292.66	302×

For runtime, Ratio denotes speedup; for communication, it denotes the reduction factor.

Table 7: Online and offline runtime: CoSecRAG vs. PRAG.

N	Online runtime (s)			Offline runtime (s)		
	CoSecRAG	PRAG	Speedup	CoSecRAG	PRAG	Speedup
2^{14}	0.69	39.70	58×	1.13	1.87	2×
2^{16}	1.37	48.66	35×	2.25	6.01	3×
2^{18}	3.69	75.51	20×	5.34	22.36	4×
2^{20}	9.94	192.44	19×	10.68	89.24	8×
2^{22}	31.99	595.58	19×	25.82	410.68	16×

For PRAG at $N = 2^{22}$, we split the feature dimension into four chunks to avoid the backend's single-tensor size limit. Runtime ablations show that chunking incurs less than 1% overhead in both online and offline runtimes.

we use the same setting fixed by the retrieval-quality study, i.e., $M = B = 2^{\lceil \log_2 \sqrt{N} \rceil}$ and $L = 8$. Table 5 reports both online runtime and total runtime, where the latter includes the offline cost, as N scales.

CoSecRAG with PSCP consistently achieves clear efficiency improvements over both our unclustered secure top-K implementation and the P^{2}RAG baseline [34]. Online, CoSecRAG with PSCP is 11× to 67× faster than the unclustered variant and 17× to 81× faster than that baseline. When offline costs are included, the total-runtime speedups are 14× to 102× and 6× to 49×, respectively. It also has a protocol-level advantage over the same baseline: this stage does not interact with the query user and does not reveal intermediate information. After subtracting communication-round latency, PSCP empirically follows an $O(\sqrt{N})$ trend over the tested range because secure selection dominates its lower-cost linear extraction; the unclustered variant follows an $O(N)$ trend.

5.4 End-to-End Vector Similarity Search Performance

We finally evaluate the complete CoSecRAG protocol in both the online and offline phases. As shown in Table 6, CoSecRAG consistently outperforms P^{2}RAG [34] in both runtime and communication. The online runtime speedup grows from 20× at $N = 2^{14}$ to 99× at $N = 2^{22}$, while online communication is reduced by 37× to 43×. In the offline phase, the runtime speedup grows from 3× to 34×, and the communication reduction factor grows from 22× to 302×. This end-to-end vector similarity search improvement comes from the combined effect of IPQ-Mask for score computation and PSCP for secure top-K computation.

PRAG [57] mainly targets a multiparty setting based on Shamir secret sharing. The authors also provide a CrypTen [27] implementation that can run in a two-server additive secret sharing setting. Using this implementation, Table 7 reports end-to-end online and offline runtime. CoSecRAG is consistently faster across all tested sizes: its online speedup ranges from 19× to 58×, and its offline speedup ranges from 2× to 16×. The decreasing online speedup and increasing offline speedup arise from different bottlenecks. Online, PRAG's round latency dominates at small N but becomes less significant as linear computation and communication grow, reducing the speedup; offline, PRAG's near-linear tuple generation grows faster than CoSecRAG's PSCP-dominated preprocessing over the evaluated range, increasing the speedup, although it should eventually stabilize.

6 CONCLUSION

We presented CoSecRAG, a two-server system for private vector similarity search in RAG. IPQ-Mask avoids costly online secure multiplications in full-database scoring. PSCP optimizes the retrieval pipeline by moving cluster pruning after scoring, reducing the secure top-K candidate set and avoiding private extraction of high-dimensional document embeddings. Together, they protect query and database embeddings while addressing the main costs of prior secret-sharing-based systems.

Experiments show that CoSecRAG maintains strong retrieval quality and improves efficiency. Compared with P^{2}RAG, its online scoring and secure top-K computation are up to 181× and 81× faster, respectively. End to end, CoSecRAG is up to 99× faster than P^{2}RAG and 19×–58× faster than PRAG online. These results demonstrate the scalability of CoSecRAG for private RAG retrieval.

REFERENCES

[1] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. 2023. Privformer: Privacy-preserving transformer with mpc. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 392–410.

[2] Amina Bassit and Vishnu Boddeti. 2025. SecureRAG: End-to-End Secure Retrieval-Augmented Generation. In *The Second Workshop on GenAI for Health: Potential, Trust, and Policy Compliance*. https://openreview.net/forum?id=njAyzNEaCd

[3] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology – CRYPTO '91*. 420–432.

[4] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptol.* 13, 1 (2000), 143–202. https://doi.org/10.1007/S001459910006

[5] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M. Sadegh Riazi. 2020. SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2111–2128. https://www.usenix.org/conference/usenixsecurity20/presentation/chen-hao

[6] Ke Cheng, Yuheng Xia, Anxiao Song, Jiaxuan Fu, Wenjie Qu, Yulong Shen, and Jiaheng Zhang. 2025. Mosformer: Maliciously Secure Three-Party Inference Framework for Large Transformers. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security* (Taipei, Taiwan) *(CCS '25)*. Association for Computing Machinery, New York, NY, USA, 4124–4138. https://doi.org/10.1145/3719027.3765028

[7] Yihang Cheng, Lan Zhang, Junyang Wang, Mu Yuan, and Yunhao Yao. 2025. RemoteRAG: A Privacy-Preserving LLM Cloud RAG Service. In *Findings of the Association for Computational Linguistics: ACL 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 3820–3837. https://doi.org/10.18653/v1/2025.findings-acl.197

[8] Cohere Team. 2023. Introducing Embed v3. Cohere Blog. https://cohere.com/blog/introducing-embed-v3 Accessed: 2026-05-26.

[9] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation

[10] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, Wen-Guang Chen, et al. 2025. Puma: Secure inference of llama-7b in five minutes. *Security and Safety* 4 (2025), 2025014.

[11] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. 2024. SIGMA: Secure GPT Inference with Function Secret Sharing. *Proc. Priv. Enhancing Technol.* 2024, 4 (2024), 61–79. https://doi.org/10.56553/POPETS-2024-0107

[12] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private Inference on Transformers. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 15718–15731. https://proceedings.neurips.cc/paper_files/paper/2022/file/64e2449d74f84e5b1a5c96ba7b3d308e-Paper-Conference.pdf

[13] Arshia Hemmat, Matin Moqadas, Ali Mamanpoosh, Amirmasoud Rismanchian, and Afsaneh Fatemi. 2025. VAGUE-Gate: Plug-and-Play Local-Privacy Shield for Retrieval-Augmented Generation. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (Eds.). The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Mumbai, India, 3715–3730. https://doi.org/10.18653/v1/2025.ijcnlp-long.194

[14] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[15] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 113–122.

[16] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Jiawen Zhang, Wen-jie Lu, Cheng Hong, and Kui Ren. 2026. CipherGPT: Secure Two-Party GPT Inference. *IEEE Transactions on Dependable and Secure Computing* 23, 3 (2026), 6748–6762. https://doi.org/10.1109/TDSC.2026.3667722

[17] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*. Springer, 145–161.

[18] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Trans. Mach. Learn. Res.* 2022 (2022). https://openreview.net/forum?id=jKN1pXi7b0

[19] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[20] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[21] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* 55, 12, Article 248 (March 2023), 38 pages. https://doi.org/10.1145/3571730

[22] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[23] Rishi Kalra, Zekun Wu, Ayesha Gulley, Airlie Hilliard, Xin Guan, Adriano Koshiyama, and Philip Colin Treleaven. 2024. HyPA-RAG: A Hybrid Parameter Adaptive Retrieval-Augmented Generation System for AI Legal and Policy Applications. In *Proceedings of the 1st Workshop on Customizable NLP: Progress and Challenges in Customizing NLP for a Domain, Application, Group, or Individual (CustomNLP4U)*, Sachin Kumar, Vidhisha Balachandran, Chan Young Park, Weijia Shi, Shirley Anugrah Hayati, Yulia Tsvetkov, Noah Smith, Hannaneh Hajishirzi, Dongyeop Kang, and David Jurgens (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 237–256. https://doi.org/10.18653/v1/2024.customnlp4u-1.18

[24] Ehsan Kamalloo, Nandan Thakur, Carlos Lassance, Xueguang Ma, Jheng-Hong Yang, and Jimmy Lin. 2024. Resources for Brewing BEIR: Reproducible Reference Models and Statistical Analyses. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024, Washington DC, USA, July 14-18, 2024*, Grace Hui Yang, Hongning Wang, Sam Han, Claudia Hauff, Guido Zuccon, and Yi Zhang (Eds.). ACM, 1431–1440. https://doi.org/10.1145/3626772.3657862

[25] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*. 6769–6781.

[26] Andes Y. L. Kei and Sherman S. M. Chow. 2025. SHAFT: Secure, Handy, Accurate, and Fast Transformer Inference. *IACR Cryptol. ePrint Arch.* 2025 (2025), 2324. https://eprint.iacr.org/2025/2324

[27] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. CrypTen: Secure Multi-Party Computation Meets Machine Learning. *CoRR* abs/2109.00984 (2021). arXiv:2109.00984 https://arxiv.org/abs/2109.00984

[28] Tatsuki Koga, Ruihan Wu, Zhiyuan Zhang, and Kamalika Chaudhuri. 2024. Privacy-preserving retrieval-augmented generation with differential privacy. *arXiv preprint arXiv:2412.04697* (2024).

[29] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[30] Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. 2023. MPCFORMER: FAST, PERFORMANT AND PRIVATE TRANSFORMER INFERENCE WITH MPC. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=CWmvjOEhgH-

[31] Zhengyi Li, Kang Yang, Jin Tan, Wen-jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, and Jingwen Leng. 2024. Nimbus: Secure and Efficient Two-Party Inference for Transformers. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 21572–21600. https://doi.org/10.52202/079017-0680

[32] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. 2025. BumbleBee: Secure Two-party Inference Framework for Large Transformers. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/bumblebee-secure-two-party-inference-framework-for-large-transformers/

[33] Microsoft. 2026. Overview of the Microsoft 365 Copilot Retrieval API. https://learn.microsoft.com/en-us/microsoft-365/copilot/extensibility/api/ai-services/retrieval/overview. Accessed: July 23, 2026.

[34] Yulong Ming, Mingyue Wang, Jijia Yang, Jie Xu, Zihan Wu, Cong Wang, and Xiaohua Jia. 2026. P^2RAG: Efficient Privacy-Preserving RAG Service Supporting Arbitrary Top-k Retrieval. *arXiv preprint arXiv:2603.14778* (2026).

[35] Junki Mori, Kazuya Kakizaki, Taiki Miyagawa, and Jun Sakuma. 2026. Differentially Private Synthetic Text Generation for Retrieval-Augmented Generation (RAG). In *Findings of the Association for Computational Linguistics, ACL 2026,*

San Diego, California, United States, July 2-7, 2026, Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (Eds.). Association for Computational Linguistics, 1216–1235. https://doi.org/10.18653/V1/2026.FINDINGS-ACL.62

[36] Syed Shariyar Murtaza, Yifan Nie, Elias Avan, Utkarsh Soni, Wanyu Liao, Adam Carnegie, Cyril John Mathias, Junlin Jiang, and Eugene Wen. 2025. Implementing Retrieval Augmented Generation Technique on Unstructured and Structured Data Sources in a Call Center of a Large Financial Institution. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, Weizhu Chen, Yi Yang, Mohammad Kachuee, and Xue-Yong Fu (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 598–606. https://doi.org/10.18653/v1/2025.naacl-industry.48

[37] OpenAI. 2024. New embedding models and API updates. OpenAI Blog. https://openai.com/index/new-embedding-models-and-api-updates/ Accessed: 2026-05-27.

[38] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *VLDB J.* 33, 5 (2024), 1591–1615. https://doi.org/10.1007/S00778-024-00864-X

[39] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*. 4753–4771. https://doi.org/10.1109/SP54263.2024.00130

[40] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[41] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[42] Peter Rindal, Lance Roy, and Lucien K. L. Ng. [n.d.]. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[43] Phillipp Schoppmann, Lennart Vogelsang, Adrià Gascón, and Borja Balle. 2020. Secure and Scalable Document Similarity on Distributed Databases: Differential Privacy to the Rescue. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 209–229. https://doi.org/10.2478/POPETS-2020-0024

[44] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. 2022. Private Approximate Nearest Neighbor Search with Sublinear Communication. In *2022 IEEE Symposium on Security and Privacy (SP)*. 911–929. https://doi.org/10.1109/SP46214.2022.9833702

[45] Hayim Shaul, Dan Feldman, and Daniela Rus. 2020. Secure k-ish Nearest Neighbors Classifier. *Proc. Priv. Enhancing Technol.* 2020, 3 (2020), 42–61. https://doi.org/10.2478/POPETS-2020-0045

[46] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Joaquin Vanschoren and Sai-Kit Yeung (Eds.). https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html

[47] Baiqiang Wang, Qian Lou, Mengxin Zheng, and Dongfang Zhao. 2025. PIR-RAG: A System for Private Information Retrieval in Retrieval-Augmented Generation. arXiv:2509.21325 [cs.IR] https://arxiv.org/abs/2509.21325

[48] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (2021), 1964–1978. https://doi.org/10.14778/3476249.3476255

[49] Yongqin Wang, G Edward Suh, Wenjie Xiong, Benjamin Lefaudeux, Brian Knott, Murali Annavaram, and Hsien-Hsin S Lee. 2022. Characterization of MPC-based private inference for transformer-based models. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 187–197. https://doi.org/10.1109/ISPASS55109.2022.00025

[50] Ruihan Wu, Erchi Wang, Zhiyuan Zhang, and Yu-Xiang Wang. 2025. Private-RAG: Answering Multiple Queries with LLMs while Keeping Your Data Private. *arXiv preprint arXiv:2511.07637* (2025).

[51] Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. 2024. Benchmarking Retrieval-Augmented Generation for Medicine. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 6233–6251. https://doi.org/10.18653/v1/2024.findings-acl.372

[52] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 162–167.

[53] Dixi Yao and Tian Li. 2025. Private retrieval augmented generation with random projection. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*. https://openreview.net/forum?id=5DfhoxRPXh

[54] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 220–250.

[55] Pengcheng Zhou, Yinglun Feng, and Zhongliang Yang. 2025. Provably secure retrieval-augmented generation. *arXiv preprint arXiv:2508.01084* (2025).

[56] Martin Zuber and Renaud Sirdey. 2021. Efficient homomorphic evaluation of k-NN classifiers. *Proc. Priv. Enhancing Technol.* 2021, 2 (2021), 111–129. https://doi.org/10.2478/POPETS-2021-0020

[57] Guy Zyskind, Tobin South, and Alex Pentland. 2024. Don't forget private retrieval: distributed private similarity search for large language models. In *Proceedings of the Fifth Workshop on Privacy in Natural Language Processing*. 7–19.

A FORMAL SECURITY ANALYSIS

This appendix gives the full simulation argument for Theorem 4.2. We use the ideal functionality $\mathcal{F}_{\mathsf{ClusteredTopK}}$ from Definition 4.1.

Throughout this appendix, we consider well-formed executions satisfying the fixed-point range condition in Section 2, the sentinel condition stated after Algorithm 4, and the input-domain condition in Definition 4.1.

Whenever a plaintext value is initially shared, or an ideal subfunctionality returns a secret-shared value, the notation $[\![x]\!]$ denotes an additive sharing generated with fresh uniform randomness: one share is sampled uniformly over the corresponding ring domain and the other share is fixed by reconstruction. Sharing randomness used for distinct such sharings is independent. Shares derived by local computation retain the correlations prescribed by the protocol. Persistent setup values are shared once and their shares remain unchanged across all query invocations.

A.1 Leakage and Security Definition

Definition A.1 (Public leakage). The setup leakage is

$$\mathcal{L}_{\mathsf{setup}} = \mathsf{pp} = \big(N, d, M, B, L, K, \ell, f, \mathsf{MIN}, \mathsf{layout}\big).$$

For each query, the protocol additionally leaks that an invocation occurred:

$$\mathcal{L}_{\mathsf{query}} = \big(\mathsf{invocation}\big).$$

For a multi-query execution, $\mathcal{L}_{\mathsf{query}}$ denotes the corresponding public sequence of invocation events, including their number and timing. No selected cluster, candidate identity, score, result identity, or data-dependent message length is included in the leakage.

For a corruption set $\mathcal{T}$, let

$$\mathsf{View}^{\Pi}_{\mathcal{T}}\big(\mathcal{I}, \vec{q}; \mathcal{L}_{\mathsf{query}}\big)$$

denote the joint view of the corrupted parties. The view contains their inputs, internal randomness, local states, received messages, and outputs.

Definition A.2 (Simulation-based security). A protocol Π securely realizes $\mathcal{F}_{\mathsf{ClusteredTopK}}$ for a corruption set $\mathcal{T}$ if there exists a probabilistic polynomial-time simulator $\mathsf{Sim}_{\mathcal{T}}$ such that

$$\mathsf{View}^{\Pi}_{\mathcal{T}}\big(\mathcal{I}, \vec{q}; \mathcal{L}_{\mathsf{query}}\big) \approx_c \mathsf{Sim}_{\mathcal{T}}\left(1^{\lambda}, \mathsf{pp}, \mathcal{L}_{\mathsf{query}}, \mathsf{in}_{\mathcal{T}}, \mathsf{out}_{\mathcal{T}}\right).$$

Here $\mathsf{in}_{\mathcal{T}}$ and $\mathsf{out}_{\mathcal{T}}$ are the inputs and outputs provided to the corrupted parties by $\mathcal{F}_{\mathsf{ClusteredTopK}}$. The simulator is not given any other database- or query-dependent value.

A.2 Correctness Relative to Clustered Retrieval

Lemma A.3 (Correctness under the stated execution conditions). *Under the well-formed execution conditions fixed above, if the underlying selection, extraction, and multiplication procedures are correct, then CoSecRAG returns exactly the output of* $\mathcal{F}_{\mathsf{ClusteredTopK}}$.

Proof. For one query, define

$$\mathbf{Z} = \begin{bmatrix} \widetilde{\mathbf{X}} \\ \mathbf{C} \end{bmatrix}.$$

The data owner samples

$$\vec{a} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d$$

and computes the joint response $\mathbf{Z}\vec{a}$. The servers open

$$\vec{e} = \vec{q} - \vec{a}.$$

Their joint score shares reconstruct to

$$\begin{aligned} \mathbf{Z}\vec{a} + \mathbf{Z}\vec{e} &= \mathbf{Z}\vec{a} + \mathbf{Z}(\vec{q} - \vec{a}) \\ &= \mathbf{Z}\vec{q}. \end{aligned}$$

The first MB entries are the document scores $\widetilde{\mathbf{X}}\vec{q}$, and the remaining M entries are the centroid scores $\mathbf{C}\vec{q}$. Therefore, the first secure selection obtains exactly the L clusters selected by $\mathcal{F}_{\mathsf{ClusteredTopK}}$. Batched extraction returns the score, identifier, and validity records aligned with those clusters.

For every extracted padded record, dummy suppression computes

$$s' = \mathsf{MIN} + v(s - \mathsf{MIN}).$$

If $v = 1$, then $s' = s$. If $v = 0$, then $s' = \mathsf{MIN}$. Under the assumptions of the lemma, dummy records cannot enter the final top-K. The final selection and identifier extraction therefore return exactly the identifier set prescribed by $\mathcal{F}_{\mathsf{ClusteredTopK}}$. □

A.3 Simulation of IPQ-Mask Scoring

Lemma A.4 (One-time masked opening). *For every fixed query* $\vec{q} \in \mathbb{Z}_{2^\ell}^d$, *if*

$$\vec{a} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d,$$

then

$$\vec{e} = \vec{q} - \vec{a}$$

is uniformly distributed over $\mathbb{Z}_{2^\ell}^d$ *and is statistically independent of* $\vec{q}$.

Proof. For every $\vec{e}_0 \in \mathbb{Z}_{2^\ell}^d$,

$$\begin{aligned} \Pr[\vec{e} = \vec{e}_0] &= \Pr[\vec{a} = \vec{q} - \vec{e}_0] \\ &= 2^{-\ell d}. \end{aligned}$$

This probability is independent of $\vec{q}$. □

Lemma A.5 (Stateful simulation of one server's scoring view). *Suppose that the persistent database and metadata values, and all per-query query, mask, and mask-response values, are additively shared according to the convention above. For any polynomial-length public query schedule, the joint scoring view of either one server can be perfectly simulated from the public setup state and the query schedule.*

Proof. Fix a corrupted server $\mathcal{S}_b$, where $b \in \{0, 1\}$, and let

$$\mathbf{Z} = (\widetilde{\mathbf{X}}^\top, \mathbf{C}^\top)^\top.$$

At setup, the simulator samples

$$\mathbf{Z}_b, \qquad (\mathsf{ID})_b, \qquad \vec{v}_b$$

uniformly over their corresponding domains and stores them as persistent state. These values have exactly the distribution of the corrupted server's setup shares. In particular, although the underlying plaintext values may be correlated, their individual shares are mutually independent because the corresponding sharings use independent randomness.

Also at setup, for every scheduled query invocation j, the simulator independently samples and stores the corrupted server's shares

$$\vec{a}_b^{(j)} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d, \qquad (\mathbf{Z}\vec{a}^{(j)})_b \xleftarrow{\$} \mathbb{Z}_{2^\ell}^{MB+M}.$$

For invocation j, it samples the freshly supplied query share

$$\vec{q}_b^{(j)} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d$$

and retrieves the corresponding unused mask and mask-response shares from its simulated setup state. It computes the corrupted server's local difference share

$$\vec{e}_b^{(j)} = \vec{q}_b^{(j)} - \vec{a}_b^{(j)}.$$

It then samples

$$\widehat{\vec{e}}^{(j)} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d$$

and defines the simulated message from the other server as

$$\widehat{\vec{e}}_{1-b}^{(j)} = \widehat{\vec{e}}^{(j)} - \vec{e}_b^{(j)}.$$

By Lemma A.4, conditioned on the entire preceding view, $\widehat{\vec{e}}^{(j)}$ has exactly the distribution of the opened masked difference in the real execution.

Finally, the simulator computes the corrupted server's score share as

$$\widehat{\mathbf{t}}_b^{(j)} = (\mathbf{Z}\vec{a}^{(j)})_b + \mathbf{Z}_b\widehat{\vec{e}}^{(j)}.$$

It splits this value into document-score and centroid-score shares exactly as in the real protocol. The persistent shares are sampled only once, whereas all query-dependent shares are freshly sampled for each invocation. Therefore, the simulated joint view over all scheduled queries is identically distributed to the real scoring view. □

A.4 Proof of Theorem 4.2

Proof. We construct a stateful simulator for every corruption set listed in Theorem 4.2. Correctness follows from Lemma A.3. All simulators receive the public query schedule $\mathcal{L}_{\mathsf{query}}$ and are activated according to its invocation events. Since this schedule contains no query value or database-dependent information, it determines only when the per-invocation simulations below are activated. Persistent setup state is sampled once and retained throughout the execution.

Corrupted data owner. The data owner receives no online message. Given its ideal input $\mathcal{I}$ and the public parameters, the simulator uses the fixed clustered index specified by $\mathcal{I}$ and the public padded layout. It samples the randomness used to share the padded database, centroids, identifiers, and validity bits. For each scheduled query, it samples an independent one-time mask, computes the corresponding joint response, and samples the outgoing additive shares according to the real setup algorithm. These values have exactly the distribution of the data owner's real setup view. Since the data owner receives no online message, its view contains no query-dependent information.

Corrupted query user. For each scheduled invocation, the simulator is given the query $\vec{q}$ and the ideal output

$$\mathsf{ID}_K^\star = \mathcal{F}_{\mathsf{ClusteredTopK}}(\mathcal{I}, \vec{q}).$$

It samples

$$\vec{q}_0 \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d, \qquad \vec{q}_1 = \vec{q} - \vec{q}_0,$$

and samples a random additive sharing of the output identifiers:

$$(\mathsf{ID}_K^\star)_0 \xleftarrow{\$} \mathbb{Z}_{2^\ell}^K, \qquad (\mathsf{ID}_K^\star)_1 = \mathsf{ID}_K^\star - (\mathsf{ID}_K^\star)_0.$$

By the sharing convention fixed above, these values have exactly the distribution of the messages generated and received by the user in the real protocol. Applying this procedure independently to every scheduled invocation simulates the user's complete view.

Corrupted server. Fix $\mathcal{S}_b$. First replace every invocation of Π_{top}, Π_{ext}, Π_{extB}, and Π_{mult} by its ideal functionality. By the assumed semi-honest security of these protocols and the sequential composition theorem [4], this hybrid is computationally indistinguishable from the real execution.

In the hybrid execution, the simulator samples the persistent setup shares once and generates the complete multi-query scoring view using Lemma A.5. For every scheduled query, it then invokes the simulators of the underlying selection, extraction, and multiplication protocols on the corrupted server's simulated local inputs and outputs. All intermediate outputs received by one server are fresh additive shares sampled with the randomness prescribed by the corresponding ideal subfunctionality.

Within each invocation, the numbers and dimensions of all subprotocol calls depend only on pp, while the number and timing of invocations are given by $\mathcal{L}_{\mathsf{query}}$. In particular, every query selects L secret-shared cluster selectors, extracts L blocks of length B, and processes exactly LB candidate records. The simulator can therefore reproduce the complete communication shape without knowing the query, selected clusters, candidates, scores, or output identifiers.

Corrupted query user and server. Fix the coalition $\mathcal{T} = \{\mathcal{U}, \mathcal{S}_b\}$. At setup, the simulator samples the corrupted server's persistent database and metadata shares once, exactly as in the corrupted-server case. For every scheduled invocation j, it also samples and stores a uniform mask share and a uniform mask-response share for the corrupted server, matching the per-query preprocessing received during the real setup.

For each scheduled invocation j, the simulator is given the query $\vec{q}^{(j)}$ and the ideal output $\mathsf{ID}_K^{\star,(j)}$. It first samples the user's query-sharing randomness:

$$\vec{q}_0^{(j)} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d, \qquad \vec{q}_1^{(j)} = \vec{q}^{(j)} - \vec{q}_0^{(j)}.$$

It independently samples

$$\widehat{\vec{e}}^{(j)} \xleftarrow{\$} \mathbb{Z}_{2^\ell}^d$$

and defines

$$\widehat{\vec{a}}^{(j)} = \vec{q}^{(j)} - \widehat{\vec{e}}^{(j)}.$$

Let $\widehat{\vec{a}}_b^{(j)}$ be the corrupted server's mask share stored during the simulated setup, and define the missing share by reconstruction. The simulator then computes

$$\widehat{\vec{e}}_b^{(j)} = \vec{q}_b^{(j)} - \widehat{\vec{a}}_b^{(j)}, \qquad \widehat{\vec{e}}_{1-b}^{(j)} = \widehat{\vec{e}}^{(j)} - \widehat{\vec{e}}_b^{(j)}.$$

Consequently, the corrupted server's local difference share and the message received from the other server are consistent with the opened value $\widehat{\vec{e}}^{(j)}$.

Using the stored mask-response share, the simulator computes the corrupted server's score shares with the same local formula as in the real protocol. It next samples

$$(\mathsf{ID}_K^{\star,(j)})_b \xleftarrow{\$} \mathbb{Z}_{2^\ell}^K$$

and defines

$$(\mathsf{ID}_K^{\star,(j)})_{1-b} = \mathsf{ID}_K^{\star,(j)} - (\mathsf{ID}_K^{\star,(j)})_b.$$

The first value is used consistently as the corrupted server's local output share and as the corresponding message received by the user. The simulator then invokes the simulators of the underlying ideal subfunctionalities on the corrupted server's simulated local inputs and outputs, including this fixed final output share.

Although the coalition can reconstruct $\widehat{\vec{a}}^{(j)} = \vec{q}^{(j)} - \widehat{\vec{e}}^{(j)}$, its persistent database shares and fresh mask-response shares remain uniformly distributed because their sharing randomness is independent. Thus the simulated joint view is computationally indistinguishable from the real view and reveals no database information beyond the output supplied by $\mathcal{F}_{\mathsf{ClusteredTopK}}$.

Combining the above simulations proves the theorem. □

A.5 Multiple Queries and Mask Reuse

The simulators above are stateful. They sample the persistent database, centroid, identifier, and validity shares once during setup and reuse the same shares for every query invocation. During setup, they also sample one fresh mask sharing and one fresh sharing of the corresponding mask response for every scheduled invocation. Online, they sample fresh query shares and fresh ideal-subfunctionality output shares for each query. Hence the proof extends to every polynomial-length public schedule for which setup provides one unused IPQ-Mask instance per invocation. Conditioned on all previous transcripts, the next opened value

$$\vec{e}^{(j)} = \vec{q}^{(j)} - \vec{a}^{(j)}$$

remains uniform and independent of the next query.

The one-time condition is necessary. If a mask is reused for two queries, then the opened values reveal

$$\vec{e}^{(1)} - \vec{e}^{(2)} = \vec{q}^{(1)} - \vec{q}^{(2)}.$$

The protocol therefore marks every IPQ-Mask instance as consumed after one query, including queries that terminate before returning an output.