Practical and Verifiable Encrypted Vector Search for Retrieval-Augmented Generation

Xiangyu Hui
xiangyu.hui.1@student.unimelb.edu.au
The University of Melbourne
Australia

Xingliang Yuan
xingliang.yuan@unimelb.edu.au
The University of Melbourne
Australia

Olga Ohrimenko
oohrimenko@unimelb.edu.au
The University of Melbourne
Australia

Sid Chi-Kin Chau
sid.chau@acm.org
CSIRO Data61
Australia

Abstract

Retrieval-augmented generation (RAG) systems critically depend on a vector-retrieval stage that selects relevant documents from a large embedding database. When this stage is outsourced to a RAG-as-a-Service provider, query embeddings can reveal sensitive user intent, the outsourced index can leak proprietary corpus information, and a malicious provider can silently manipulate retrieval results. This motivates privacy-preserving verifiable retrieval. The client must be able to confirm that the returned top-k identifiers were produced by faithfully executing an agreed approximate nearest neighbor (ANN) algorithm on a committed encrypted index, while leaking no non-public database information beyond the functional baseline induced by the returned top-k identifiers.

We present VeriANN, the *first* encrypted ANN retrieval framework, to our knowledge, that simultaneously achieves *query privacy*, *database confidentiality*, and *verifiability of retrieval results* against malicious servers, under a two-server non-colluding trust model. VeriANN couples distributed-point-function–based PIR over locality-sensitive hashing indexes with authenticated garbled circuits, so that the entire top-k pipeline—bucket decryption, Merkle-root reconstruction, frequency counting, and top-k selection—is executed obliviously and with end-to-end integrity. Making this integration practical requires three new techniques: (i) a sort-based hierarchical oblivious frequency-counting algorithm that enables a distance-free post-processing stage, reducing top-k aggregation from quadratic to quasi-linear complexity; (ii) an end-to-end authenticated verification design that binds the full retrieval pipeline against selective-failure attacks while reducing client-side verification to a single hash check against the published Merkle root; and (iii) a modular state-pool design with an authenticated state-transfer mechanism that dynamically composes precomputed garbled states across query parameters while preserving cross-circuit verifiability. On million-scale corpora, VeriANN achieves second-scale end-to-end latency with KB-scale client-to-server communication, while adding minimal online overhead over a non-verifiable baseline.

Keywords

Encrypted vector search, query privacy, RAG verifiability

1 Introduction

Retrieval-Augmented Generation (RAG) grounds large language models in external knowledge bases through a vector-retrieval stage that is increasingly outsourced to RAG-as-a-Service (RaaS) providers such as Amazon Bedrock [1], Azure OpenAI [39], and Google Vertex AI [22]. In a typical RAG pipeline, both documents and user queries are represented as high-dimensional embeddings, and the retriever selects documents whose embeddings are closest to the query embedding. Since modern corpora may contain millions of vectors, this retrieval step is usually implemented as approximate nearest-neighbor (ANN) search rather than exhaustive similarity search. This outsourcing exposes three security gaps that are not jointly addressed by current RAG deployments. First, query embeddings are dense fingerprints of user intent: observing them can reveal sensitive information [40, 45]. Second, the vector index encodes proprietary corpus content, so access to the index or intermediate retrieval states may leak confidential database information [40]. Third, a malicious or compromised provider can silently substitute or omit retrieved results, steering the LLM toward attacker-chosen answers [44, 60].

A trustworthy outsourced retrieval service must therefore provide three guarantees simultaneously. *Query privacy* hides the client's query and access pattern from retrieval servers. *Database confidentiality* bounds what clients (and any server colluding with them) learn about the corpus to the functional baseline induced by the returned top-k identifiers. *verifiability of retrieval results* lets the client reject any result that was not faithfully produced by the prescribed ANN algorithm on the committed index.

Existing work covers only subsets. As summarized in Table 1, prior private ANN systems each leave at least one property out. Tiptoe [23] hides the query but return plaintext candidates to the client. Compass [59] achieves verifiability via Path ORAM but assumes the client owns the database and holds the ORAM key, ruling out multi-client RaaS. Panther [37] composes PIR, GC, and HE for retrieval, but its heterogeneous composition is proved only in the semi-honest model and does not provide end-to-end integrity against malicious servers. Perco [43] pioneers two-server DPF–PIR + LSH in the semi-honest setting, but its first-non-empty-bucket strategy lacks a secure ranking stage and shows limited accuracy scalability on high-dimensional text-embedding corpora. The challenge is to make private retrieval, secure oblivious ranking, and client-side verification compose end-to-end under a malicious adversary, without paying the prohibitive cost.

This work. We present VeriANN, the *first* encrypted ANN framework, to our knowledge, that simultaneously provides query privacy, database confidentiality, and verifiability of retrieval results under the standard two-server non-colluding model. We highlight our contributions as follows.

First, we design an end-to-end authentication layer for outsourced ANN retrieval. Rather than only authenticating individual PIR answers, VeriANN extends authenticity across the PIR–MPC–client boundary to the final top-k output. After DPF-PIR retrieves encrypted bucket shares, the buckets are decrypted and authenticated inside AuthGC by in-circuit Merkle-root reconstruction. The reconstruction serve as verification anchors, bound to the ranked candidate list and a client-sampled query tag in a single commitment. The query tag prevents the accept/abort bit from becoming a selective-failure oracle, while client verification reduces to a single hash check.

Second, we introduce a distance-free secure ranking method that turns LSH collision frequency into an MPC-friendly similarity signal. Given the authenticated bucket contents, VeriANN ranks candidate identifiers by their cross-table collision counts rather than by high-dimensional distances. This avoids costly in-circuit distance computation while keeping secure top-k practical and preserving the accuracy scalability of LSH.

Third, we introduce an authenticated state-transfer mechanism for modular pre-computation, allowing the expensive post-processing circuit and lightweight top-k selection circuits to be preprocessed separately and dynamically composed at query time without breaking end-to-end authenticity.

Finally, we formalize and prove query privacy, database confidentiality, and client verifiability for VeriANN, covering client–server collusion and selective-failure behavior.

On million-scale corpora, VeriANN achieves practical online latency: 1.87–10.89 s across SIFT and MS-MARCO, while keeping client-to-server communication to only 12–47 KB and sustaining high retrieval accuracy at moderate index settings. To isolate the cost of verifiability, we compare VeriANN against a non-verifiable variant of itself and find that verifiability adds minimal overhead in both online latency and communication. Against private ANN scheme *Perco* [43], VeriANN matches end-to-end latency within 1.5× on MS-MARCO while reducing client–server communication by up to 3.8×, and additionally provides verifiability, dynamic top-k, and scalable retrieval accuracy on large text corpora such as MS-MARCO. Against *Compass* [59], VeriANN stays within a comparable latency range while reducing client–server communication by three to four orders of magnitude, and additionally provides dataset confidentiality and multi-client support.

2 Related Work

Encrypted similarity search. Early work constructs searchable encryption schemes for similarity search by treating LSH hashes as keywords [34, 55, 56]. Persistent leakage-abuse attacks against searchable encryption [7, 52] have motivated a shift toward stronger cryptographic primitives that aim to eliminate search and access-pattern leakage, including the primitives used by private ANN retrieval systems.

Table 1: Comparison of private ANN retrieval systems.

System	Query Privacy	Dataset Conf.	Verifiability	Distance Free*	Multi-Client‡
Tiptoe [23]	✓	✗	✗	✗	✓
Perco [43]	✓	✓	✗	✓†	✓
Panther [37]	✓	✓	✗	✗	✓
Compass [59]	✓	✗	✓	✗	✗
VeriANN (ours)	✓	✓	✓	✓	✓

* Distance-free means secure ranking avoids distance evaluation, which otherwise incurs high MPC/storage cost or requires communication-heavy client interaction.
† Perco is distance-free only via a first-non-empty-bucket heuristic that ties output size to bucket size at build time; no dynamic top-k, false positives unfiltered.
‡ Multi-client means one outsourced index can serve multiple independent clients in a shared retrieval service.

Private ANN retrieval systems. We compare VeriANN more closely with four representative systems summarized in Table 1. Compass [59] traverses an HNSW graph over Path ORAM, but assumes a single-client setting where the client owns the database, holds the ORAM key, and caches HNSW-related metadata locally. Each query is an interactive graph traversal: the client repeatedly fetches and re-encrypts ORAM paths while evaluating candidate neighbors. This incurs high client–server communication, prevents a single outsourced index from serving independent clients, and provides no confidentiality against the querying client. Tiptoe [23] combines HE with clustering over a vector database, returning the nearest cluster to the client for local ranking; its follow-up Wally [5] and the graph-based Pacmann [58] share the same assumption and therefore offer neither dataset confidentiality nor verifiability. Panther [37], building on SANNS [8], combines PIR, secret sharing, GC, and HE over clustering-based ANN in the single-server setting, achieving practical semi-honest performance but requiring the client to participate in every stage, and offering no path to end-to-end verifiability: extending their heterogeneous mix of primitives to malicious security would require re-proving each subprotocol and chaining authenticated outputs across primitive boundaries. Perco [43] pioneers the two-server DPF-PIR + LSH paradigm for private ANN search with sublinear communication. Its first-non-empty-bucket heuristic avoids distance computation, but fixes the bucket size to the output size at index-build time, precludes dynamic top-k queries, and leaves false positives unfiltered. Consequently, on MS-MARCO, scaling from 16 tables / 2 probes to 64 tables / 8 probes improves accuracy only from 56.1% to 57.8%, whereas VeriANN exceeds 95% under the same parameters. Suppressing such spurious collisions would require lookup tables of size at least 2^{35}, making commitments and practical verifiability prohibitive. Song et al. [46] later shrink this size by reintroducing a distance-computation stage, but do not address malicious servers. Very recently, zkRAG [27] explores a different point in the design space, using zero-knowledge proofs to attest retrieval correctness under a threat model that focuses on dataset confidentiality with public queries.

Adversarial attacks and defenses in RAG. A parallel line studies adversarial attacks at the RAG application layer [16, 21, 54, 60], where attackers inject crafted content into the database or pipeline

to mislead the LLM backend. Defenses based on certified robustness [51] and ML-based guardrails [31] have been proposed. VeriANN is complementary: it adds an encryption-and-verification layer at the retrieval stage, orthogonal to these application-layer defenses. Other emerging threats including membership inference [18, 41] and denial-of-service [44] are out of scope.

3 Preliminaries

Vector Retrieval in RAG. Retrieval-Augmented Generation (RAG) enhances large language models by grounding their outputs on external knowledge bases [19, 36]. In a standard RAG pipeline, a user query q is first encoded into a dense embedding vector $\mathbf{v}_q$ using a pretrained encoder. The system then retrieves relevant documents by comparing $\mathbf{v}_q$ against a corpus of precomputed document embeddings $\{\mathbf{v}_i\}$ stored in a vector database, typically via Approximate Nearest Neighbor (ANN) search.

Formally, let $D = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \mathbb{R}^d$ be a database of n vectors. A data object $o \in D$ is a *c-approximate nearest neighbor* (c-ANN) of a query $\mathbf{v}_q$ if $\|o - \mathbf{v}_q\| \leq c \cdot \|o^* - \mathbf{v}_q\|$, where o^* is the exact nearest neighbor of $\mathbf{v}_q$ in D and $c \geq 1$ is the approximation ratio. The c-approximate k-NN ((c,k)-ANN) problem is to find k data objects that are each a c-ANN of $\mathbf{v}_q$. The retriever returns the top-k approximately most relevant document identifiers, after which the corresponding documents may be further refined by a reranker or relevance judge before being provided to the LLM.

Locality-Sensitive Hashing. Locality-Sensitive Hashing (LSH) [25] is a classical approach for approximate nearest neighbor search. At a high level, LSH probabilistically groups nearby vectors into the same hash buckets, enabling efficient candidate retrieval without exhaustive search. Formally, a hash family $\mathcal{H}$ is (R, cR, p_1, p_2)-sensitive if for any $x, y \in \mathbb{R}^d$, $\Pr_{h\sim\mathcal{H}}[h(x) = h(y)] \geq p_1$ when $\mathrm{dist}(x,y) \leq R$, and $\Pr_{h\sim\mathcal{H}}[h(x) = h(y)] \leq p_2$ when $\mathrm{dist}(x,y) \geq cR$, where $c > 1$ and $p_1 > p_2$.

In practice, LSH constructs l hash tables using independently sampled hash functions $\{h_m\}_{m=0}^{l-1}$, where each table maps vectors to buckets. Increasing l reduces the failure probability δ. To improve recall, multi-probe LSH [38] further examines multiple candidate buckets per table. For a query vector $\mathbf{v}_q$, the m-th table produces a sequence of probe buckets $(b_{m,0}, \ldots, b_{m,p-1}) \leftarrow \mathsf{MultiProbe}(h_m, \mathbf{v}_q, p)$. Typically, LSH uses bucket probing to generate a candidate set, which is then refined by a post-processing step that ranks or filters the candidates to produce the top-k result.

DPF-based Two-Server PIR. Private Information Retrieval (PIR) enables a client to retrieve a database record without revealing the accessed index to the servers. In this work, we consider a two-server PIR setting, where the database is replicated across two non-colluding servers.

Let $D = (x_0, \ldots, x_{N-1})$ denote a database replicated at two servers $\mathcal{S}_0$ and $\mathcal{S}_1$, where $[N] = \{0, \ldots, N-1\}$. Given an index $i \in [N]$, the client wishes to retrieve x_i without revealing i to either server. A common construction of two-server PIR relies on Distributed Point Functions (DPF). A DPF allows the client to encode a point function $\phi_i : [N] \to \{0,1\}$, where $\phi_i(i) = 1$ and $\phi_i(j) = 0$ for all $j \neq i$, into two correlated keys, such that each key individually reveals no information about i. Formally, a two-server DPF-based PIR scheme consists of the following algorithms:

- $(dk^{(0)}, dk^{(1)}) \leftarrow \mathsf{DPF.Gen}(i, 1^\lambda)$: Given an index i and security parameter λ, the client generates two DPF keys encoding the point function ϕ_i.
- $\phi_i^{(j)}(t) \leftarrow \mathsf{DPF.Eval}(dk^{(j)}, t)$: Given a DPF key $dk^{(j)}$ and an index $t \in [N]$, server $\mathcal{S}_b$ locally evaluates its share of the point function
- $a^{(b)} \leftarrow \mathsf{Answer}(dk^{(b)}, D)$: Server $\mathcal{S}_b$ computes an answer share

$$a_b \leftarrow \bigoplus_{t=0}^{N-1} \Big(\mathsf{DPF.Eval}(dk^{(b)}, t) \cdot \mathbf{x}[t]\Big),$$

  where the $\cdot$ denotes selection of the database element.
- $x_i \leftarrow \mathsf{Reconstruct}(a_0, a_1)$: The client reconstructs the requested record by computing $x_i = a_0 \oplus a_1$.

Merkle Tree. A Merkle tree is a cryptographic data structure that enables compact commitments to a set of values for efficient membership verification. Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a cryptographic hash function. Given a sequence of values $\mathbf{x} = (x_1, \ldots, x_n)$, a Merkle tree is constructed by hashing each value to form the leaf nodes and recursively hashing pairs of child nodes until a single root hash is obtained.Formally, a Merkle commitment scheme consists of the following algorithms:

- $(rt, \mathsf{Tree}) \leftarrow \mathsf{Merkle.Build}(\mathbf{x})$: Given a sequence of values $\mathbf{x}$, outputs a Merkle root rt that serves as a commitment to $\mathbf{x}$, together with the corresponding tree structure.
- $\pi_i \leftarrow \mathsf{Merkle.ProveInclude}(\mathsf{Tree}, i)$: Given the Merkle tree and an index $i \in [n]$, outputs a Merkle inclusion proof π_i for value $\mathbf{x}_i$.
- $rt^* \leftarrow \mathsf{Merkle.Reconstruct}(\pi, \mathbf{x}_i)$: Given a Merkle inclusion proof π_i, and the corresponding leaf value $\mathbf{x}_i$, reconstructs the Merkle root rt^* by iteratively hashing from the leaf to the root.

A Merkle inclusion proof π_i consists of the sibling hashes on the authentication path from the leaf of $\mathbf{x}_i$ to the root, along with bits indicating each sibling's position. To verify, compute $rt^* \leftarrow \mathsf{Merkle.Reconstruct}(\pi_i, \mathbf{x}_i)$ and accept if $rt^* = rt$.

Authenticated Garbling (AuthGC). Authenticated garbling [47] extends garbled circuits [53] to achieve maliciously secure two-party computation. Given private inputs $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$ held by two parties, the protocol enables them to jointly compute $\mathbf{y} = c(\mathbf{x}^{(0)}, \mathbf{x}^{(1)})$ for a circuit c, without revealing their inputs. The garbler encodes the circuit into a garbled form, which the evaluator can evaluate to obtain an encoded representation of the output.

AuthGC proceeds in two phases: an *offline phase* and an *online phase*. In the offline phase, both parties generate input-independent state for the circuit. In the online phase, each party provides its private input, and the evaluator evaluates the garbled circuit to obtain Yao shares of the output.

We model authenticated garbling as a black-box primitive below:

- $st^{(p)} \leftarrow \mathsf{AuthGC.Init}(p, c, pp)$: executed in the offline phase to generate input-independent state $st^{(p)}$ for circuit c, given public parameters pp.
- $[\![\mathbf{y}]\!]_Y^{(p)} \leftarrow \mathsf{AuthGC.Online}(p, st^{(p)}, \mathbf{x}^{(p)})$: executed in the online phase, where each party provides its private input $\mathbf{x}^{(p)}$, and the parties obtain Yao shares $[\![\mathbf{y}]\!]_Y^{(p)}$ of the output.

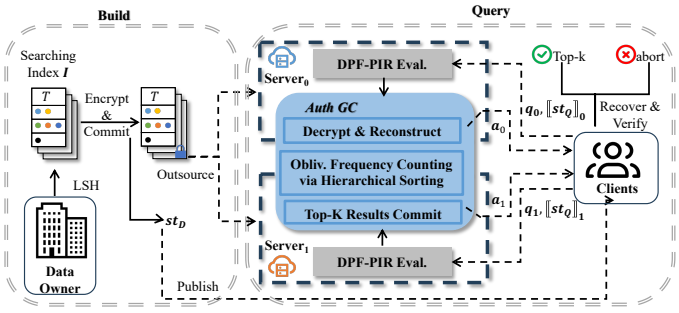


Figure 1: VeriANN system overview. The data owner outsources an encrypted LSH index with a public commitment st_D to two non-colluding servers. A client submits query shares $(q_j, [[st_Q]]_j)$; the servers jointly run DPF-PIR and AuthGC to produce output shares α_j, which the client recovers and verifies against st_D and st_Q

The resulting Yao shares can be locally converted into Boolean shares $[\![y]\!]_B$ using the standard Yao-to-Boolean (Y2B) conversion from the ABY framework [15], which incurs negligible computation and communication overhead.

4 System Model

4.1 Overview

VeriANN involves a *data owner* with a private knowledge database, *clients* issuing ANN queries, and *two non-colluding servers* that store and search an encrypted index on their behalf (Figure 1). The protocol has four stages:

Build$(D) \rightarrow (I, st_D)$. The data owner constructs an encrypted searchable index I over embedding vectors D and a public commitment st_D to it. The two servers receive I; st_D is published to clients.

Query$(v, k) \rightarrow (q^{(0)}, q^{(1)}, [\![st_Q]\!]^{(0)}, [\![st_Q]\!]^{(1)})$. For an embedding v and a chosen k, the client generates a query $q^{(j)}$ for each server and a one-time secret state st_Q that binds the subsequent verification to this query.

Answer$(I, q^{(j)}, [\![st_Q]\!]^{(j)}) \rightarrow a^{(j)}$. Each server $\mathcal{S}_j$ jointly with the other server runs the secure retrieval, post-processing protocol over I to produce a share $a^{(j)}$ of the result.

Recover$(a^{(0)}, a^{(1)}, st_Q, st_D) \rightarrow \{\mathbf{id}_{\text{top-}k}, \perp\}$. The client reconstructs the top-k identifiers from the two shares and verifies their consistency with st_D using st_Q, accepting the result or aborting.

The complete protocol specification is given in §6.5.

Scope. VeriANN returns the top-k identifiers $\mathbf{id}_{\text{top-}k}$; fetching the actual documents is delegated to the underlying encrypted storage layer, which can be instantiated with an oblivious backend (e.g., Authenticated PIR[9] over encrypted blobs keyed by identifier). Subsequent accesses may leak through that layer unless it is oblivious, designing such a storage layer is orthogonal to VeriANN.

4.2 Threat model

We consider a private and verifiable ANN retrieval service for RAG built upon the two-server non-colluding trust model, involving a data owner, multiple clients, and two servers hosting an encrypted search index.

We adopt the standard two-server non-colluding model used by prior distributed-trust systems [9–11, 14, 35]: security holds as long as at least one of the two servers remains honest, while the other may be fully compromised and arbitrarily deviate from the protocol. The model is instantiated in production deployments such as DAP/Prio aggregation [20] and Apple's iCloud Private Relay [4], where the two parties reside in independent trust domains and violating the assumption requires cross-domain collusion [13].

Malicious retrieval server. A malicious server is computationally bounded and may arbitrarily deviate from the prescribed ANN retrieval protocol. In particular, it may intentionally return responses that differ from those of the true database, e.g., by modifying a small number of rows or tampering with intermediate results, and then observe whether the client accepts or rejects the response. Such behavior corresponds to *selective-failure attacks* [9, 24, 32, 47]. It may further attempt to manipulate retrieval results, inject adversarial identifiers, or infer sensitive information about client queries and access patterns from observed protocol transcripts.

Client–malicious server collusion. We allow at most one of the two retrieval servers to collude with a client in an attempt to break VeriANN's security guarantees. This models realistic RAG deployments in which the retrieval service is accessible to many users, and a server can register as a client and issue a query.

4.3 Security and Correctness Guarantees.

VeriANN provides the following guarantees for private and verifiable ANN retrieval in RAG.

Query privacy. Against any single malicious server—even when colluding with arbitrary other clients—VeriANN ensures that no information about an honest client's query embedding or access pattern is revealed, even when the server observes all protocol transcripts and the client's accept-or-abort behavior.

Database confidentiality. VeriANN ensures that the outsourced searchable index remains confidential throughout the retrieval process. Against an adversary that corrupts the client and colludes with at most one of the two servers, the per-query information learned is strictly limited to the *functional baseline leakage* (Definition 7.1) intrinsic to the (c, k)-ANN functionality: the top-k identifiers $\mathbf{id}_{\text{top-}k}$ themselves, plus the implicit information that the corresponding vectors are returned as (c, k)-ANN results.

Verifiability of retrieval results. Clients can efficiently verify that the returned top-k identifiers are correctly computed from the committed vector index according to the prescribed ANN retrieval algorithm. All returned identifiers must originate from the authenticated searchable index and correspond to the top-k identifiers produced by a faithful execution of the prescribed ANN retrieval procedure. Any malicious deviation by the server, including result manipulation, index tampering, or selective omission, is detected with overwhelming probability and causes the client to abort.

We give informal statements in §7 and defer the formal definitions and proofs to Appendix F.

Non-goals. As in prior private ANN search [37, 43, 46], our security guarantees are stated *per query*; we do not defend against attacks that combine information across adaptive queries, including implicit

Protocol 1: Strawman ANN Retrieval

Public Parameters. Security parameter λ, LSH family $\mathcal{H} = \{h_m\}_{m=0}^{L-1}$,bucket bound s; number of probes p.

Build (Data Owner).

- For each table T_m, $m \in [l]$
  - For each embedding v_i, compute $b \leftarrow h_m(\mathbf{v}_i)$ and insert $\mathbf{v}_i$ into bucket B_b.
  - For each bucket B:
    - If B is non-empty, normalize it to size s.
    - Encrypt $\widetilde{B} \leftarrow \mathsf{AES}(B, sk_m, b)$ and insert $\widetilde{B}$ into T_m.
  - Establish $(rt_m, Tree_m)$ over all encrypted buckets via Merkle.Build.
- Upload $I = \{T_m, Tree_m\}_{m\in[l]}$ to servers, publish $\{rt_m, sk_m\}_{m\in[l]}$ to the clients.

Query (Client).

- Compute $\{b_{m,n}\}_{n\in p} \leftarrow \mathsf{MultiProbe}(h_m, \mathbf{v}_q, p)$, $m \in [l]$.
- For each $m \in [l]$ and $n \in [p]$, generate $(dk_{m,n}^{(0)}, dk_{m,n}^{(1)}) \leftarrow \mathsf{DPF.Gen}(b_{m,n}, 1)$.
- Send $\{dk_{m,n}^{(j)}\}_{m\in[l],n\in[p]}$ to server $\mathcal{S}_j$ for $j \in \{0, 1\}$.

Answer (Servers).

- For each $m \in [l]$ and $n \in [p]$, each server $\mathcal{S}_j$ computes $a_{m,n}^{(j)} \leftarrow \bigoplus_{i=0}^{N_T-1} \mathsf{DPF.Eval}(dk_{m,n}^{(j)}, i) \cdot (T_{m,i} \| \mathsf{Merkle.ProveInclude}(Tree_m, i))$
- Each server $\mathcal{S}_j$ returns $\{a_{m,n}^{(j)}\}$ to the client.

Post-processing (Client).

- For each $m \in [l]$ and $n \in [p]$,
  - Reconstruct the answer $(\widetilde{B}_{m,n}, \pi_{m,n}) = a_{m,n}^{(0)} \oplus a_{m,n}^{(1)}$.
  - Decrypt $B_{m,n} \leftarrow \mathsf{AES}(B, sk_m, b)$.
  - Verify $rt_m \stackrel{?}{=} \mathsf{Merkle.Reconstruct}(\pi_{m,n}, \|B_{m,n})$.
- Select the top-k identifiers with highest frequency across all retrieved buckets.

Protocol 1: Strawman construction.

knowledge extraction [50], attribute inference via result differencing [57], and membership inference across adaptive queries [3, 41]. We also exclude server availability (denial-of-service), network-level metadata leakage (query timing, traffic volume), and side channels. The data owner is assumed honest at Build time. We leave mitigation of those privacy inference attacks in the encrypted RAG domain as future work.

5 From a Strawman Construction

We begin by developing an intuitive strawman construction that motivates the design choices in VeriANN. Inspired by prior work on LSH-based similarity search [34, 43, 46, 55, 56], we adopt LSH as the retrieval paradigm. Concretely, the data owner constructs l hash tables from independently sampled hash functions $\{h_m\}_{m=0}^{l-1}$; for each table T_m, document identifiers are mapped to buckets according to h_m. To hide bucket occupancy, all buckets are normalized to size s (under-full buckets are padded with nearest neighbors of the bucket centroid), encrypted under a per-table symmetric key, and outsourced to the two servers as the encrypted index I.

At query time, the client computes multi-probe LSH to determine a set of target bucket indices across l tables. For each target bucket, the client encodes its index as a DPF key pair [6] and sends one key to each server. Each server evaluates its key over the entire table and returns an XOR share of the selected encrypted bucket. The client reconstructs and decrypts the bucket from the two shares. Crucially, from either server's view, the evaluation touches every row of the table uniformly, so no information about which bucket was queried is revealed.

To guard against malicious servers, we adopt the two-server authenticated PIR framework of [9], treating each LSH bucket as a data row. For each hash table, a Merkle tree is built over its encrypted buckets, with the root published as a table commitment. During retrieval, each server returns the encrypted bucket together with a Merkle inclusion proof, allowing the client to verify that the retrieved content matches the committed index. This construction is selective-failure-attack free [24]. The full construction is summarized in Protocol 1.

Limitations. The strawman scheme fails in two respects. First, revealing the plaintext contents of all l probed buckets to the client already exceeds the functional baseline leakage permitted by our security definition, let alone the case where the client colludes with a corrupted server to exfiltrate the entire encrypted index out-of-band and decrypt it offline using the provided key. In short, distributing decryption keys to the client violates database confidentiality.

Second, returning plaintext bucket contents to the client incurs prohibitive communication overhead on large, high-dimensional corpora. Each bucket stores s embedding vectors of dimension dim, and each entry carries a Merkle inclusion proof with $O(\log N_T)$ sibling hashes, each 256 bits. Across all l probes, the total client-to-server communication is $l \cdot p \cdot \left(s \cdot \dim + 256 \cdot \log N_T\right)$ bits, which is impractical for RAG-scale deployments.

Therefore, a desired protocol must reveal only the final $\mathbf{id}_{\text{top-}k}$, keep per-query client-visible leakage within the functional baseline, and still allow the client to verify that the result was faithfully computed from the committed index. These requirements motivate the design of VeriANN.

6 Towards VeriANN Searching

The two requirements identified in Section 5 give rise to two interdependent design challenges. Keeping intermediate bucket contents hidden from the client is necessary to preserve database confidentiality and ensure that the query process leaks no more than the functional baseline leakage. At the same time, producing a verifiable result demands *client verifiability*: the client must be able to confirm, without seeing any bucket contents, that $\mathbf{id}_{\text{top-}k}$ is the faithful output of the prescribed ANN algorithm on the committed index. Producing that result securely then requires an efficient *secure post-processing algorithm* that computes the top-k jointly over the servers' private inputs, leaking nothing beyond the final output.

These two challenges are interdependent: the post-processing algorithm must be structured so that client verifiability is achievable on top of it, and the verifiability mechanism must remain compatible with the computation's structure. We therefore address client verifiability first (§ 6.1), establish the cryptographic foundation it

requires, and then build the secure post-processing algorithm on top of it (§ 6.2, § 6.3).

6.1 Achieve Client Verifiability

Challenge I: client verifiability cannot be achieved by MPC alone. A natural starting point is to move post-processing inside a maliciously secure two-party computation (2PC) protocol [12, 30, 47]. This intuition, however, does not work in two respects.

1) Input consistency is not guaranteed. Malicious 2PC ensures that both servers faithfully execute the prescribed circuit *on whatever inputs they provide*, but it does not enforce that those inputs are the actual bucket contents retrieved by the PIR stage. A malicious server may substitute tampered contents before feeding them into the 2PC, while the circuit itself executes correctly. The mutual-enforcement mechanism is blind to input provenance: it detects deviations inside the circuit, but has no handle on what arrives at the circuit boundary.

2) Mutual enforcement does not extend to the client. Malicious 2PC provides a *bilateral* guarantee between the two servers; the client is not a participant and receives only the final output. Whatever consistency the servers enforce on each other is not transferable to a third party: the client has no cryptographic handle to distinguish a faithful execution from an arbitrary one consistent with the returned output. Public verifiability requires a witness available to an external verifier, which mutual enforcement by construction does not produce.

Together, these two gaps define the absence of client verifiability: the first leaves the circuit's inputs unbound to the PIR stage, while the second leaves its execution unverifiable to the client.

Solution I: Merkle root reconstruction inside MPC as a verification anchor. The key observation is that a Merkle root is a cryptographic commitment to the bucket contents. Collision resistance ensures that no adversary can produce a consistent root from tampered inputs. This makes the Merkle root a natural verification anchor, i.e., a compact, binding digest that the client can check without seeing the underlying records.

Concretely, VeriANN performs bucket decryption and Merkle root reconstruction inside a maliciously secure two-party computation protocol. Since both operations are naturally expressed as Boolean circuits, i.e., AES-128 for bucket decryption and SHA-256 for iterative Merkle root reconstruction, we instantiate this computation using authenticated garbling [47], which is an efficient maliciously secure 2PC protocol for Boolean-circuit evaluation. Reconstruction is performed directly over the decrypted bucket contents used as circuit inputs, so any tampering at the input boundary propagates into an inconsistent root. The per-bucket Merkle root reconstruction is formalized in Algorithm 1.

Since all p probe buckets in the same LSH table are authenticated under one Merkle tree, their reconstructed roots $rt_{m,1}, \ldots, rt_{m,p}$ must agree. Taking $rt_{m,1}$ as the reference, the circuit uses an oblivious equality-and-select gadget: if all p reconstructed roots agree, it outputs their common value; otherwise, it outputs a dummy root.

The circuit then aggregates the per-table roots as $rt_{\text{agg}} \leftarrow H(rt_1 \| \cdots \| rt_l)$, which binds the post-processing computation to the authenticated retrieval inputs. A natural first step is to compute the verification commitment $cm \leftarrow H\big(rt_{\text{agg}} \,\|\, \mathbf{id}_{\text{top-}k}\big)$, where $\mathbf{id}_{\text{top-}k}$ is the top-k identifier vector produced by the secure post-processing circuit. The client accepts only if the reconstructed commitment is consistent with the published index commitment and the returned top-k identifiers.

Challenge II: Selective-failure attack surface. The commitment cm defined above appears to bind the servers to the computed result: any forgery seemingly requires a hash collision against rt_I. Yet the verification scheme still exposes two *selective-failure* vectors that bypass this barrier.

1) Caused by low-entropy outputs. After honest circuit evaluation, $\mathcal{S}_0$ holds $(\llbracket \mathbf{id}_{\text{top-}k} \rrbracket^{(0)}, \llbracket cm \rrbracket^{(0)})$. Instead of forwarding them, it picks a target $\mathbf{id}'_{\text{top-}k}$, *guesses* the true result as $\hat{\mathbf{id}}_{\text{top-}k}$, and submits

$$\llbracket \mathbf{id}_{\text{top-}k} \rrbracket'^{(0)} := \llbracket \mathbf{id}_{\text{top-}k} \rrbracket^{(0)} \oplus \hat{\mathbf{id}}_{\text{top-}k} \oplus \mathbf{id}'_{\text{top-}k},$$
$$\llbracket cm \rrbracket'^{(0)} := \llbracket cm \rrbracket^{(0)} \oplus H(rt_I \,\|\, \hat{\mathbf{id}}_{\text{top-}k}) \oplus H(rt_I \,\|\, \mathbf{id}'_{\text{top-}k}).$$

If $\hat{\mathbf{id}}_{\text{top-}k} = \mathbf{id}_{\text{top-}k}$, the client reconstructs $\mathbf{id}'_{\text{top-}k}$ and the check passes. The success probability is not negligible: top-k identifiers are drawn from a bounded corpus with highly non-uniform distribution, and a semantically informed adversary can narrow the plausible set for any given query. On an incorrect guess, the abort itself leaks $\hat{\mathbf{id}}_{\text{top-}k} \neq \mathbf{id}_{\text{top-}k}$, so repeated queries progressively shrink the uncertainty over the true result.

2) Caused by bucket content hiding. A second selective-failure channel arises at the circuit input. Each bucket ships with a Merkle proof against st_D, but the proof authenticates the bucket in isolation, with no binding to its position. A malicious server can therefore replace one authenticated bucket with another: after linear aggregation, the result fails verification when the client's query missed the substituted slot (abort), and reconstructs to a valid authenticated bucket when it hit (accept). The abort/accept bit thus leaks the access pattern, and the accepting case delivers the wrong bucket, breaking correctness as well as privacy. A standard fix is to embed the bucket index inside each bucket and let the client check it. This fix is unavailable here because bucket contents are not revealed to the client. See Appendix E for the full construction.

1) Client-sampled query state. For each query, the client samples a fresh randomness $r \leftarrow \{0,1\}^\lambda$ and assembles the per-query state $\text{st}_Q = (r, \mathbf{b})$, where $\mathbf{b} = \{b_{m,n}\}$ are the multi-probe LSH bucket indices. The client XOR-secret-shares st_Q as $(\llbracket \text{st}_Q \rrbracket^{(0)}, \llbracket \text{st}_Q \rrbracket^{(1)})$, sending one share to each server. Both components remain information-theoretically hidden from any single server: r as fresh randomness, and $\mathbf{b}$ as part of the same XOR sharing.

2) Binding bucket index and decryption key to leaves. At index construction, the data owner incorporates each LSH bucket index b and the per-table key $sk_{T,m}$ into its Merkle leaf as $H(b \,\|\, \widetilde{B} \,\|\, sk_{T,m})$, so that both the position of a bucket and the decryption key under which it was encrypted are cryptographically tied to its contents. Inside the circuit, $\mathbf{b}$ is reconstructed from $\llbracket \text{st}_Q \rrbracket$ and $sk_{T,m}$ from $\llbracket sk_{T,m} \rrbracket$, both participating in leaf-hash re-derivation during Merkle verification. Any ciphertext substitution that moves a bucket to a different index, or any in-circuit decryption performed under a tampered key, fails verification except under a hash collision.

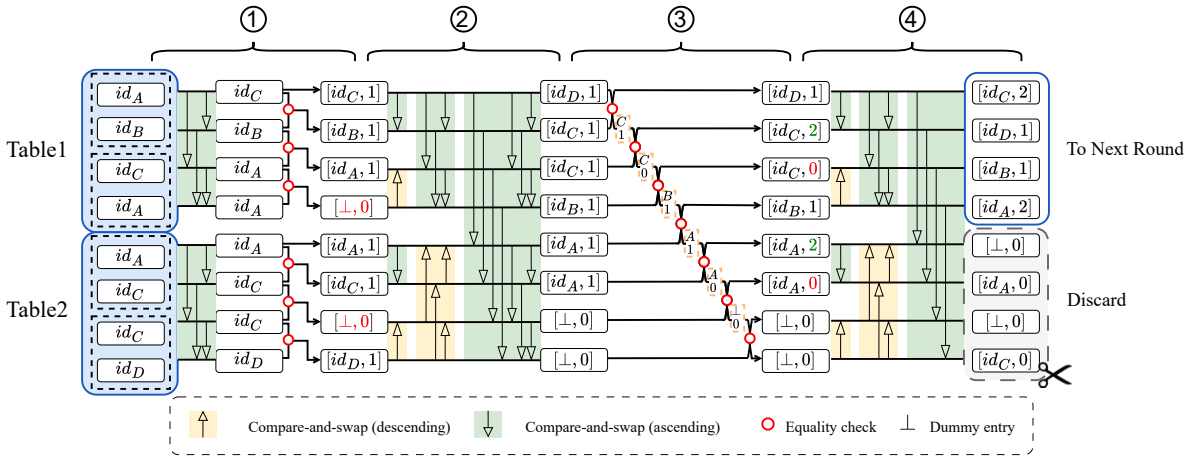


Figure 2: Sorting-based hierarchical frequency counting. An example with 2 tables, 2 probes per table, and bucket size 2. ① each table's candidates are bitonic-sorted by identifier and adjacent within-table duplicates are replaced by $(\perp, 0)$; ② the two per-table lists are concatenated and sorted by identifier to align cross-table duplicates; ③ an oblivious linear scan aggregates duplicate counts into one slot and zeros out the others; ④ retains the top-w candidates by frequency for the next round.

3) Entropy-bound commitment. The final commitment computed inside the circuit is $cm \leftarrow H(\mathbf{id}_{\text{top-}k} \,\|\, \mathbf{b} \,\|\, r \,\|\, rt_I)$[1], absorbing the unpredictable randomness r in cm. A malicious server attempting to forge $[\![cm]\!]'^{(0)}$ for an arbitrary $\mathbf{id}'_{\text{top-}k}$ would have to predict r without access to r, which succeeds with probability at most $2^{-\lambda}$. Since r is freshly sampled per query, an abort on one query reveals nothing reusable for subsequent queries, collapsing the selective-failure channel.

6.2 Distance-Free Secure Post-Processing

Challenge III: high-dimensional vector distance computation is prohibitively expensive in MPC. A straightforward post-processing approach is to compute distances between the query embedding and each retrieved candidate inside the garbled circuit, then sort by distance to select the top-k results. However, this is prohibitively expensive for two reasons. First, since the index is stored as ciphertext, identifier-based deduplication is impossible, so every bucket entry must carry its full embedding vector; across l tables this inflates the index size by orders of magnitude. Second, each distance computation over 384-dimensional vectors incurs substantial non-XOR gate cost in garbled circuits. Figure 7 reports the cost for a modest candidate set of 16 vectors: the circuit reaches 2.517 GB (34.05M AND gates), and this is *before* sorting. Since the cost scales linearly in both the number of candidates and the embedding dimension, distance-based ranking inside authenticated garbling is impractical at realistic retrieval scales.

Solution III: secure distance-free ranking via oblivious frequency counting. We build on the key observation from C2LSH [17]: vectors that are closer in embedding space tend to collide across more LSH hash tables. This suggests that collision frequency across retrieved buckets serves as an effective proxy for embedding similarity, enabling ranking without any distance computation. Concretely, we select the top-k identifiers with the highest occurrence frequencies across tables, after deduplicating identifiers within each table. This reduces post-processing to a frequency-counting problem over integer identifiers. This distance-free formulation brings two advantages. 1) It is far more amenable to garbled-circuit evaluation than floating-point arithmetic over high-dimensional vectors. 2) Since only identifiers need to be stored per bucket entry, the index size is reduced by orders of magnitude compared to storing full embedding vectors. We empirically validate this design in Section D.3: frequency-based ranking attains accuracy above 90% at moderate index settings and scales monotonically with the number of tables, probes, and bucket size, confirming that collision frequency is a faithful similarity proxy in practice.

6.3 Efficient Oblivious Frequency Counting via Hierarchical Sorting

Challenge IV: oblivious frequency counting is expensive. Frequency counting is straightforward in plaintext: one maintains a hash map and increments the counter for each observed identifier. Inside a garbled circuit, however, this approach is entirely unavailable. Garbled circuits do not permit secret-dependent control flow or secret-dependent memory access. All computation must be *oblivious*, meaning the sequence of operations must be fixed and independent of the secret inputs. Conditionally updating a counter at a secret-indexed location is therefore infeasible. To preserve obliviousness, one must instead compare each identifier against every other and update counts in a data-independent manner, resulting in $O(n^2)$ comparisons for an input stream of size n. As the number of retrieved bucket entries grows with the number of hash tables and probes, this naive approach becomes quickly impractical.

[1]We later refine this commitment to bind through an intermediate query tag $tag_q = H(\mathbf{b} \,\|\, r)$ for compatibility with the modular state-pool design (§C); the entropy argument carries over since tag_q inherits r's freshness.

Solution IV: oblivious sorting-based hierarchical frequency aggregation. We exploit a structural observation about LSH retrieval. Within a single bucket, the s retrieved identifiers are duplicate-free by construction. Across the p probes of one table, padding may introduce duplicates. After per-table deduplication, multiplicity collapses to at most 1 within any table and at most 2 across any two merged tables. This bounded duplication enables an efficient hierarchical sort-based pipeline: each table is first deduplicated in isolation, then per-table lists are merged pairwise to accumulate cross-table collision counts—the C2LSH similarity signal.

Per-table deduplication. Within a single table, we concatenate the p candidate lists of size s from a single table into an array **id** of length $w = p \cdot s$ and apply an oblivious bitonic sort, bringing duplicate identifiers into adjacent positions. A single linear scan then sweeps the sorted array: at each position j, an oblivious comparison flag b_{dup} is set if $\mathbf{id}[j]$ matches any of the preceding $p-1$ entries. Using b_{dup} as a multiplexer, the scan replaces $\mathbf{id}[j]$ with a dummy $\perp$ and sets $\mathbf{freq}[j]$ to 0 when a duplicate is detected, and otherwise leaves $\mathbf{id}[j]$ unchanged with $\mathbf{freq}[j] = 1$ (Consolidate, Algorithm 2).

Cross-table hierarchical aggregation. Given two per-table lists of size w, we concatenate them into arrays **id** and **freq** of length $2w$, apply an oblivious bitonic sort keyed on **id**, and perform an oblivious linear scan: for each position j, if $\mathbf{id}[j] = \mathbf{id}[j-1]$ and $\mathbf{id}[j]$ is not a dummy, we accumulate $\mathbf{freq}[j]$ into $\mathbf{freq}[j-1]$ and zero out $\mathbf{freq}[j]$, collapsing each duplicate pair into a single slot with its aggregated count. After consolidation, we apply a second oblivious sort keyed on **freq** to rank candidates by frequency (MergeCount, Algorithm 3).

To incorporate all l tables, we organize computation as a hierarchical pairwise reduction: in each round, adjacent candidate lists are merged using the above procedure, halving the number of lists. After $\log l$ rounds, a single candidate list of size w remains, from which the top-k identifiers are extracted as the final output. The overall complexity of this hierarchical aggregation is $O(n \log^2 w)$, a significant asymptotic improvement over the $O(n^2)$ naive oblivious approach, yielding an order-of-magnitude reduction in circuit size (Section D.2). Accuracy still scales monotonically with l, p, and s, with the gap to direct counting bounded by a single parameter step (Section D.3). Figure 2 illustrates the four-step procedure on a small example with $l = 2$, $p = 2$, $s = 2$.

Optimizations to the bitonic sort. The three bitonic sorts above each admit a tailored optimization that reduces compare-and-swap operations. First, since each probe already returns candidates sorted by identifier, which we preserve on disk, deduplication begins with a p-way merge over the p sorted runs rather than a full bitonic sort. Second, each cross-table merge retains only the top-w of $2w$ aggregated candidates, so it suffices to ensure that every upper-half item has frequency no smaller than any lower-half item; the final bitonic merge stage can thus be omitted, saving $w \log_2(2w)$ compare-and-swap operations per round. Third, the final top-k extraction uses only the smallest stage whose bitonic subsequence length exceeds k, since later stages would only reorder discarded items.

6.4 Modular Pre-computation with Dynamic Top-k

Different deployments may require different output sizes. VeriANN therefore supports a build-time set $\mathcal{K}$ of admissible top-k values, with per-client subsets assigned by trust level. Since the functional leakage of any (c, k)-ANN protocol grows with k (Def. 7.1), $\mathcal{K}$ also serves as a per-query leakage budget. Unlike LSH-based private ANN systems such as Perco [43], where output size is tied to bucket size and changing k requires rebuilding the index, VeriANN decouples k from index construction: a single index with bucket size s supports all $k \le s$, since k affects only the final selection step.

This decoupling enables splitting the pipeline into a post-processing circuit PP, which outputs a consolidated size-w list $(\mathbf{id}, \mathbf{freq})$, and a lightweight selection circuit TK_k that extracts the top-k identifiers. Both modules are compiled and preprocessed separately as AuthGC circuits, with one-time states drawn from parameterized state pools at query time; § C describes the corresponding mechanism.

The remaining challenge is to preserve verifiability across the circuit boundary. AuthGC authenticates wires only within one circuit, so naively passing PP's output to TK_k would allow a malicious server to replace $(\mathbf{id}, \mathbf{freq})$ while keeping the honest Merkle root rt, causing TK_k to commit to a forged top-k result. To prevent this, VeriANN uses an authenticated state-transfer mechanism based on a query tag $tag_q = H(\mathbf{b} \,\|\, r)$, where $\mathbf{b}$ and r are the query-state components introduced in §6.1. At the end of PP, the circuit outputs $rt' = rt \oplus H(\mathbf{id} \,\|\, \mathbf{freq} \,\|\, tag_q)$ together with $(\mathbf{id}, \mathbf{freq}, tag_q)$. The selection circuit recovers $rt = rt' \oplus H(\mathbf{id} \,\|\, \mathbf{freq} \,\|\, tag_q)$ before computing the final commitment $cm = H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, rt)$. Any boundary tampering with $(\mathbf{id}, \mathbf{freq})$ changes the recovered root unless the adversary predicts tag_q, which is hidden from any single server by XOR sharing and fresh per query. The client checks cm against the published index commitment, so cross-circuit substitution is detected with overwhelming probability.

6.5 Put Things Together

Protocol 3 specifies the complete VeriANN pipeline, extending the strawman in §5 (Protocol 1) with the blue steps. Build is a one-time data-owner setup, while Query, Answer, and Recover run per query. Rationale is deferred to §6.1–§6.4.

Build. The data owner constructs the encrypted LSH index as in the strawman of §5: each non-empty bucket is normalized to size s and encrypted under a per-table AES key $sk_{T,m}$. Following §6.1, the per-table Merkle tree is built over leaf hashes $H(b_{m,i} \,\|\, \widetilde{B}_{m,i} \,\|\, sk_{T,m})$, binding both the bucket index and the decryption key to the commitment. Each $sk_{T,m}$ is XOR-secret-shared across the two servers as $(\llbracket sk_{T,m} \rrbracket^{(0)}, \llbracket sk_{T,m} \rrbracket^{(1)})$ so that decryption is performed only inside the circuit, and the offline state pools are populated via authenticated garbling (§6.4, §C): $\mathsf{Pool}_{\mathsf{PP}}$ and $\mathsf{Pool}_{\mathsf{TK}}$, where PP (Circuit 1) and TK (Circuit 2) implement the post-processing and top-k selection circuits, respectively.

Query. Given query embedding $\mathbf{v}_q$, the client runs multi-probe LSH to obtain $\mathbf{b} = \{b_{m,n}\}_{m,n}$ and encode $\mathbf{b}$ into DPF keys $(dk^{(0)}_{m,n}, dk^{(1)}_{m,n}) \leftarrow \mathsf{DPF.Gen}(b_{m,n}, 1)$. It samples fresh $r \xleftarrow{\$} \{0,1\}^\lambda$ and XOR-secret-shares $st_Q = (r, \mathbf{b})$ (§6.1) as $(\llbracket st_Q \rrbracket^{(0)}, \llbracket st_Q \rrbracket^{(1)})$. The client sends $q^{(j)} = \big(k,\ \{dk^{(j)}_{m,n}\}_{m,n},\ \llbracket st_Q \rrbracket^{(j)}\big)$ to server $\mathcal{S}_j$.

Protocol 2: Secure Oblivious Top-k Retrieval

Functionality:
$\mathsf{ObliTopK}(k, \{\mathbf{a}^{(j)}_{m,n}\}, \{[\![sk_{T,m}]\!]^{(j)}\}, [\![st_Q]\!]^{(j)}) \rightarrow ([\![\mathbf{id}_{\text{top-}k}]\!]^{(j)}, [\![cm]\!]^{(j)})$
Parameters:
l: # hash tables; p: # probes per table; k: # returned items
s: bucket size; d: Merkle path length;
$w = p \cdot s$: candidate set size
Sub-circuits: $\mathsf{PP}(s, d, p, l)$, $\mathsf{TK}(k, w)$
State pools: $\mathsf{Pool}_{\mathsf{PP}}$, $\mathsf{Pool}_{\mathsf{TK}}$
Server inputs (for each $\mathcal{S}_j$, $j \in \{0,1\}$):
PIR aggregation shares: $\{\mathbf{a}^{(j)}_{m,n}\}_{m\in[l],\, n\in[p]}$
AES key shares: $\{[\![sk_{T,m}]\!]^{(j)}\}_{m\in[l]}$
Query-state share: $[\![st_Q]\!]^{(j)}$

Each server $\mathcal{S}_j$, $j \in \{0,1\}$, executes the following:
$st_{\mathsf{PP}} \leftarrow \mathsf{Pool}^{(j)}_{\mathsf{PP}}[s, d, p, l]$
$\mathbf{x}^{(j)}_{\mathsf{PP}} \leftarrow (\{\mathbf{a}^{(j)}_{m,n}\}, \{[\![sk_{T,m}]\!]^{(j)}\}, [\![st_Q]\!]^{(j)})$
$[\![\mathbf{y}_{\mathsf{PP}}]\!]^{(j)} \leftarrow \mathsf{AuthGC.Online}(j, st_{\mathsf{PP}}, \mathbf{x}^{(j)}_{\mathsf{PP}})$
// $\mathbf{y}_{\mathsf{PP}} = (\mathbf{id}, \mathbf{freq}, rt', tag_q)$ as described in §6.4
$st_{\mathsf{TK}} \leftarrow \mathsf{Pool}^{(j)}_{\mathsf{TK}}[k, w]$
$([\![\mathbf{id}_{\text{top-}k}]\!]^{(j)}, [\![cm]\!]^{(j)}) \leftarrow \mathsf{AuthGC.Online}(j, st_{\mathsf{TK}}, [\![\mathbf{y}_{\mathsf{PP}}]\!]^{(j)})$
return $([\![\mathbf{id}_{\text{top-}k}]\!]^{(j)}, [\![cm]\!]^{(j)})$

Protocol 2: Secure oblivious top-k retrieval.

Answer. Each server $\mathcal{S}_j$ evaluates its DPF keys to obtain the PIR aggregation shares $\{\mathbf{a}^{(j)}_{m,n}\}$, each carrying an XOR share of the target ciphertext bucket and its Merkle inclusion path. The two servers jointly run Protocol 2, which composes PP and TK_k over states drawn from the offline pools. In PP, the servers consume the PIR shares, AES-key shares, and query-state shares inside AuthGC to decrypt buckets, reconstruct Merkle roots, and obliviously aggregate and rank candidate frequencies. The authenticated intermediate state $(\mathbf{id}, \mathbf{freq}, rt', tag_q)$ is passed to TK_k, which extracts the top-k identifiers and computes the final commitment cm. Each server returns $([\![\mathbf{id}_{\text{top-}k}]\!]^{(j)}, [\![cm]\!]^{(j)})$ to the client.
Result recovery and verification (Recover). The client XOR-reconstructs $\mathbf{id}_{\text{top-}k}$ and cm from the two shares, recomputes $tag_q \leftarrow H(\mathbf{b} \,\|\, r)$ from the retained st_Q, and accepts iff $cm = H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, rt_I)$; otherwise it outputs $\perp$.

Protocol 3: VeriANN Retrieval

Public Parameters. Security parameter λ; LSH family $\mathcal{H} = \{h_m\}_{m=0}^{l-1}$; bucket bound s; number of probes p.

Build (Data Owner).
- For each table T_m, $m \in [l]$
  - For each embedding v_i, compute $b \leftarrow h_m(v_i)$ and insert identifier i into bucket B_b.
  - Sample $sk_{T,m} \leftarrow \{0,1\}^\lambda$ and secret-share it as $([\![sk_{T,m}]\!]^{(0)}, [\![sk_{T,m}]\!]^{(1)})$, distributing $[\![sk_{T,m}]\!]^{(j)}$ to server $\mathcal{S}_j$ for $j \in \{0,1\}$.
  - For each bucket B:
    - If B is non-empty, normalize it to size s.
    - Encrypt $\widetilde{B} \leftarrow \mathsf{AES}(B, sk_m, b)$ and insert $\widetilde{B}$ into T_m.
  - Establish $(rt_m, Tree_m)$ over all encrypted buckets via $\mathsf{Merkle.Build}()$, where the leaf hash is computed as $H(b_{m,i} \| \widetilde{B}_{m,i} \| sk_{T,m})$.
- Upload $I = \{T_m, Tree_m\}_{m\in[l]}$ to servers, publish $st_D = rt_I \leftarrow H(rt_1 \| \ldots \| rt_l)$. to the clients.

Query (Client).
- For $m \in [l]$, compute $\{b_{m,n}\}_{n\in[p]} \leftarrow \mathsf{MultiProbe}(h_m, \mathbf{v}_q, p)$.
- For each $m \in [l]$ and $n \in [p]$, generate $(dk^{(0)}_{m,n}, dk^{(1)}_{m,n}) \leftarrow \mathsf{DPF.Gen}(b_{m,n}, 1)$.
- Sample $r \leftarrow \{0,1\}^\lambda$ and assemble the query state $st_Q = (r, \mathbf{b})$ where $\mathbf{b} = \{b_{m,n}\}_{m,n}$. Secret-share it as $([\![st_Q]\!]^{(0)}, [\![st_Q]\!]^{(1)})$.
- Send $q^{(j)} = (k, \{dk^{(j)}_{m,n}\}_{m,n}, [\![st_Q]\!]^{(j)})$ to $\mathcal{S}_j$, $j \in \{0,1\}$.

Answer (Servers).
- For each $m \in [l]$ and $n \in [p]$, each server $\mathcal{S}_j$ computes $a^{(j)}_{m,n} \leftarrow \bigoplus_{i=0}^{N_T-1} \mathsf{DPF.Eval}(dk^{(j)}_{m,n}, i) \cdot (T_{m,i} \| \mathsf{Merkle.ProveInclude}(Tree_m, i))$
- The servers jointly execute ObliTopK with inputs $(k, \{a^{(j)}_{m,n}\}, \{[\![sk_{T,m}]\!]^{(j)}\}, [\![st_Q]\!]^{(j)})$.
- Each server $\mathcal{S}_j$ outputs $a^{(j)} = ([\![\mathbf{id}_{\text{top-}k}]\!]^{(j)}, [\![cm]\!]^{(j)})$ to the client.

Client Verification and Output.
- Reconstruct the results $\mathbf{id}_{\text{top-}k} \leftarrow [\![\mathbf{id}_{\text{top-}k}]\!]^{(0)} \oplus [\![\mathbf{id}_{\text{top-}k}]\!]^{(1)}$ and $cm \leftarrow [\![cm]\!]^{(0)} \oplus [\![cm]\!]^{(1)}$.
- Rebuild the query tag $tag_q \leftarrow H(\mathbf{b} \| r)$.
- Compute $cm^* \leftarrow H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, rt_I)$.
- Output $\perp$ if $cm^* \neq cm$; otherwise output $\mathbf{id}_{\text{top-}k}$.

Protocol 3: VeriANN.

7 Security Analysis

7.1 Verifiability of VeriANN

A meaningful verifiability definition for VeriANN must expose tampering at every circuit input boundary, i.e., before values are consumed by an AuthGC instance. This includes both the boundary where PIR outputs and the associated key and query-state shares enter the post-processing circuit PP (Circuit 1), and the boundary where PP's output is forwarded as input to the selection circuit TK (Circuit 2). A definition that only quantifies over a server's final answer would miss tampering that occurs upstream of AuthGC.

Informal definition. For adversary $\mathcal{A}$ corrupting one server, let y denote the client's output when $\mathcal{A}$ may arbitrarily choose its PIR aggregation share $\mathbf{a}^{*(b)}$ and the boundary share $[\![\mathbf{y}_{\mathsf{PP}}]\!]^{*(b)}$ that it forwards from PP into TK. We say VeriANN satisfies *verifiability* if $\Pr[y \notin \{\Phi(\mathbf{v}_q, D, k), \perp\}]$ is negligible, where Φ is the prescribed deterministic top-k ANN procedure executed in the clear on the committed database. The formal game and definition (Definition F.1) are deferred to Appendix F.1, together with a refined protocol syntax that makes these input-boundary checks explicit.

Theorem 7.1 (Verifiability of VeriANN). *Assume H is modelled as a random oracle and that AuthGC realises malicious-secure*

two-party computation [47, Thm. 5.1]. Then VeriANN satisfies verifiability with negligible error in the security parameter. The proof and the explicit error bound are given in Appendix F.1.

Proof sketch. The proof in Appendix F.1 bounds the adversary's success by union-bounding over four bad events; each maps directly to a design step of §6.

- *Input-boundary tampering* is reduced to a leaf-hash collision on the index commitment (Lemma F.1); this closes the input-provenance gap of §6.1.
- *Low-entropy output forgery*, corresponding to the selective-failure attack of §6.1 (II.1), is reduced to guessing the per-query randomness r, by Lemma F.2.
- *Cross-circuit boundary tampering*—the gap of §6.3—is reduced to predicting the random-oracle mask keyed by tag_q (Lemma F.3).
- *In-circuit deviation* is bounded by the malicious-secure guarantee of AuthGC applied to each of the PP and TK instances.

Each event has negligible probability, and a final commitment check on $H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, rt_I)$ at the client rules out any residual second-preimage forgery. □

7.2 Query Privacy

Query privacy must hold even when a malicious server observes its full protocol transcript and the client's accept-or-abort decision, e.g., through retry, fallback, or error handling. This accept bit is important: selective-failure attacks can otherwise turn verification failures into an oracle on the client's access pattern or retrieval result. Thus, our definition includes the accept bit in the adversary's view and requires it to be simulable without the honest query embedding $\mathbf{v}_q$.

Informal definition. For an adversary $\mathcal{A}$ corrupting one server $b \in \{0,1\}$, let $\mathsf{REAL}_{\mathcal{A},b,D,\mathbf{v}_q,\lambda}(z)$ denote the corrupted server's view and the client's accept bit $\beta = \mathbf{1}\{y \neq \perp\}$, where the auxiliary input z may contain queries, outputs, and transcripts of adversarially controlled clients. We say VeriANN satisfies *query privacy* if there exists a PPT simulator Sim that, given only the public index state and z, but not $\mathbf{v}_q$, produces an ideal distribution $\mathsf{IDEAL}_{\mathcal{A},\mathsf{Sim},D,\lambda}(z)$ such that

$$\mathsf{REAL}_{\mathcal{A},b,D,\mathbf{v}_q,\lambda}(z) \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{A},\mathsf{Sim},D,\lambda}(z).$$

The formal game is given in Definition F.2.

Theorem 7.2 (Query Privacy of VeriANN). *Assume the DPF-based two-server PIR provides single-server privacy, AuthGC is maliciously secure against one corrupted party, XOR sharing is perfectly hiding against one share-holder, and the Merkle commitment instantiated with H is binding. Then VeriANN satisfies query privacy (Definition F.2).*

Proof sketch. The proof in Appendix F.2.4 proceeds by hybrids. First, the accept bit is replaced by the outcome determined by the adversary's effective tampering: Merkle binding, authenticated state transfer, and the entropy-bound final commitment ensure that effective tampering causes abort except with negligible probability, independently of $\mathbf{v}_q$. Next, the corrupted server's share of $st_Q = (r, \mathbf{b})$ is replaced by a uniform string by XOR-sharing hiding, and its DPF query is replaced by simulated single-server view of DPF-PIR.. Finally, the PP and TK AuthGC transcripts are simulated, hiding plaintext buckets, Merkle paths, candidate lists, and frequency counters. The resulting distribution depends only on the public index state, auxiliary input z, and simulated messages, and is therefore the ideal experiment.

7.3 Dataset Confidentiality

Per the threat model of §4.2, dataset confidentiality must hold against an adversary that simultaneously corrupts the *client* and *one of the two servers*. As in prior work on private ANN search [37, 43, 46], the guarantee is stated *per query*; cross-query inference is out of scope. Using the baseline leakage framework of [43], we recall the unavoidable per-query leakage of the (c,k)-ANN functionality and then show that VeriANN matches this baseline.

Definition 7.1 (Functional Baseline Leakage for c-ANN). *Let $\mathrm{DB} = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\} \subset \mathbb{R}^d$ with maximum pairwise distance $D_{\max}$, public search radius R, and public approximation factor $c > 1$ such that $cR < D_{\max}$. For $k \geq 1$, the per-query* functional baseline leakage *of the (c,k)-ANN functionality is bounded by*

$$\mathcal{L}_{\text{fb}} \;\leq\; k\left(\underbrace{\log N}_{(1)} \;+\; \underbrace{d \cdot \log(D_{\max}/cR)}_{(2)}\right),$$

where (1) is the explicit ID leakage and (2) is the implicit positional leakage from revealing that each returned vector lies in a cR-ball around q. The bound is tight up to a constant when the k returned vectors lie in disjoint balls; the matching information-theoretic lower bound follows from a volume argument on ℓ_p-norm balls [43, Thm. 1].

Theorem 7.3 (Dataset Confidentiality of VeriANN). *Assume IND-CPA security of bucket encryption, simulation-based malicious security of AuthGC [47], and H modelled as a random oracle. Then for any single query q, VeriANN satisfies per-query dataset confidentiality (Definition F.4): the collusion adversary's view is simulable from a leakage function $\mathcal{L}(\mathrm{DB}, q)$ whose non-public database-dependent content is upper-bounded by $\mathcal{L}_{\text{fb}}$ bits per query (Definition 7.1). The formal game, the explicit $\mathcal{L}$, and the full proof are in Appendix F.3.*

Proof sketch. A four-step hybrid simulator constructs the adversary's view from $\mathcal{L}(\mathrm{DB}, q)$: (1) replace the encrypted index with encryptions of dummy values [IND-CPA]; (2) re-execute PIR.Eval on the simulated index using the adversary-chosen DPF keys [follows from (1) by IND-CPA closure]; (3) replace the AuthGC transcript via the AuthGC simulator on the joint output $(\mathbf{id}_{\text{top-}k}, cm)$ [AuthGC malicious security]; (4) hand the corrupted client $\mathbf{id}_{\text{top-}k}$ together with $\widehat{cm}$ re-derived under the simulated commitment $\widehat{st}_D$, with indistinguishability inherited from (1).

8 Evaluation

Our evaluation aims to answer the following questions:

RQ1: Scalability. How do the PIR and GC components scale with the number of hash tables, probes, bucket size, and table scale, and how much parallelism can VeriANN exploit in each?

RQ2: End-to-end efficiency. What latency does VeriANN achieve on large datasets under realistic network conditions?

Table 2: Dataset configurations used in evaluation.

Dataset	Size	Dim	ℓ	p	s	N_T
SIFT	1M	128	8	2	64	2^{23}
MS-MARCO-Sub†	2.5M	384	16	2	64	2^{23}
MS-MARCO	8.8M	384	32	2	64	2^{24}

† Constructed by taking the union of the top-500 nearest neighbors retrieved per query from MS-MARCO using the query set.

RQ3: Client communication. Is VeriANN's client-to-server communication lightweight enough for resource-constrained clients?
RQ4: Baseline comparison. How does VeriANN compare with prior private ANN systems in latency, client-to-server communication, and the overhead of its stronger security guarantees?
RQ5: Complexity and design choices. What is the asymptotic cost of each protocol phase, and what do our design choices gain, e.g., in accuracy and computation cost? Due to space constraints, we defer the analysis to Appendix D.

8.1 Experimental Setup

Implementation and environment. Implementation details, backend modifications, and the experimental environment are described in the Open Science appendix (Appendix A).
Network conditions. We evaluate two network settings—RTT = 20 ms / 5 Gbps (nearby cross-region) and RTT = 100 ms / 2 Gbps (transcontinental or cross-cloud)—both within standard managed-interconnect tiers [2].
Metric. Throughout, we report *end-to-end latency* as the online, per-query latency, including client-side query preparation and result processing. VeriANN's offline precomputation phase is query-independent and amortizable; we report it separately and exclude it from end-to-end latency.
Datasets. We evaluate VeriANN on three benchmark datasets spanning two representative regimes for RAG retrieval. SIFT [26] is an ANN benchmark of visual descriptors, widely used for evaluating similarity retrieval. MS-MARCO [42] represents the high-dimensional, web-scale text-retrieval regime that drives modern RAG deployments. To probe the regime in between without constructing a synthetic corpus, we additionally include MS-MARCO-Sub, built by taking the union of the top-500 nearest neighbors retrieved per query from MS-MARCO using its query set; this preserves the embedding distribution of MS-MARCO while reducing scale, isolating the effect of corpus size from that of dimensionality. Index configurations are summarized in Table 2, with parameters chosen based on scalability and accuracy analyses in §8.2 and §D.3.

Baselines. We compare VeriANN against four baselines. *w/o Verif.* and *Strawman* are internal ablations that isolate the cost of verifiability and dataset confidentiality, respectively. Since no existing private ANN system meets VeriANN's full security model, we additionally include the most efficient representative from each server architecture: *Compass* (single-server, ORAM-based) and *Perco* (two-server non-colluding, DPF-PIR-based).

- *w/o Verif.*: removes the client-verifiability mechanism of §6.1 and is instantiated with semi-honest GC [48].
- *Strawman*: instantiates the Strawman construction (Protocol 1, §5), which lacks dataset confidentiality.

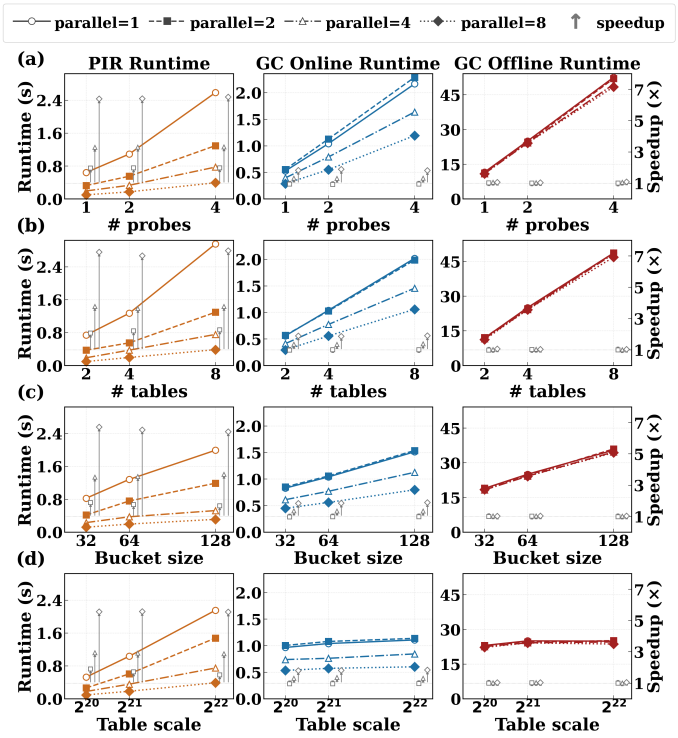


Figure 4: Scalability under system parameters.

- *Compass* [59]: a private ANN system in the *single-server* setting. It traverses an HNSW graph over Path ORAM in a single-client setting and does not provide dataset confidentiality against the querying client.
- *Perco* [43]: a private ANN system in the *two-server non-colluding* setting. Perco runs in the *semi-honest* model and provides no verifiability against malicious servers.

8.2 Scalability

We sweep four system parameters spanning the full design space, i.e., number of probes p, number of tables l, bucket size s, and per-table scale N_T—and measure their effect on the three dominant components (PIR, GC online, GC offline) under 1 to 8 concurrent threads. Figure 4 reports the results.
Effect of system parameters. Starting from the base configuration $p = 2$, $s = 64$, $\ell = 4$, $N_T = 2^{21}$, we vary one parameter at a time. PIR grows approximately linearly with p, l, N_T and s (slightly sub-linear in s due to the fixed per-bucket Merkle path overhead), GC offline grows linearly with all three parameters, as each enlarges the total circuit size. GC online follows the same trend, consistent with the analysis in §D.1. Both GC stages remain flat as N_T increases (row (d)), since N_T only marginally lengthens the in-circuit Merkle path, contributing a minimal term to the overall circuit size.
Effect of concurrency. PIR scales best, reaching up to 7× at 8 threads, exploiting the independence of DPF evaluation across (m, n) pairs. GC online scales modestly to ∼2×, as AND gates within a topological layer are dispatched in parallel on AES-NI but adjacent layers must serialize on data dependencies. GC offline shows

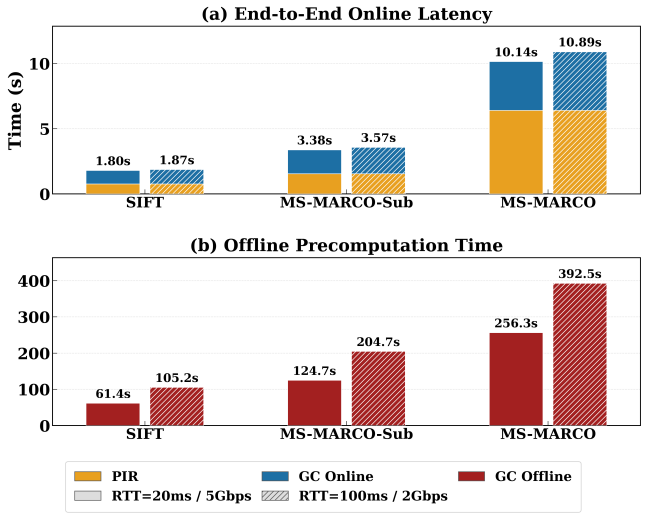


Figure 5: End-to-end latency and offline precomputation time across datasets under different network settings.

Table 3: Communication cost breakdown per query.

Dataset	C↔S	S↔S (Online)	S↔S (Offline)
SIFT	12 KB	4.62 MB	1.8 GB
MS-MARCO-Sub	23 KB	8.89 MB	3.6 GB
MS-MARCO	47 KB	17.94 MB	7.4 GB

essentially no intra-session speedup: although per-gate garbling itself parallelizes, the protocol requires garbled rows to be written to the network socket in deterministic gate order, serializing the phase at the I/O boundary. This bottleneck is sidestepped at the session level: precomputed states for distinct sessions are independent and can be garbled concurrently, amortizing the per-state offline cost across a state pool.

8.3 Performance on Benchmark Datasets

We now report VeriANN's end-to-end performance on the three datasets summarized in Table 2.

Table 3 reports the per-query communication cost. The client-to-server overhead is lightweight across all configurations, ranging from 12 KB to 47 KB, dominated by DPF keys. Server-to-server online communication scales linearly with circuit input and output size, reaching 43.28 MB on MS-MARCO.

Figure 5(a) reports the online latency. Under RTT = 20 ms / 5 Gbps, total latency is 3.89 s, 7.70 s, and 19.65 s for SIFT, MS-MARCO-Sub, and MS-MARCO, respectively, with PIR accounting for the dominant fraction in each case. Increasing the RTT to 100 ms / 2 Gbps adds only 0.18 s, 0.37 s, and 0.53 s, confirming that the online latency is computation-bound rather than network-bound.

Figure 5(b) reports the offline precomputation time, which is query-independent and fully amortized over the circuit pool. Under RTT = 20 ms / 5 Gbps, precomputation takes 68.5 s, 138.6 s, and 288.4 s for SIFT, MS-MARCO-Sub, and MS-MARCO, respectively.

Table 4: Baseline comparison on latency and client–server communication.

Method	SIFT		MS-MARCO	
	Latency (s)	Comm.	Latency (s)	Comm.
w/o Verif.	4.43	12 KB	18.88	47 KB
Strawman	0.76	26 KB	6.39	108 KB
Compass (fast)	0.10	12.2 MB	0.86	266.4 MB
Compass (slow)	0.92	12.2 MB	7.08	266.4 MB
Perco	0.48	26 KB	7.30	182 KB
VeriANN (ours)	**1.87**	**12 KB**	**10.89**	**47 KB**

8.4 Comparison with Baselines

Table 4 compares VeriANN against four baselines in end-to-end latency and client–server communication.

Compared with *w/o Verif.*, VeriANN achieves lower online latency on both SIFT (1.87 s vs. 4.43 s) and MS-MARCO (10.89 s vs. 18.88 s), despite executing a larger circuit.[2] This is because VeriANN benefits from authenticated garbling's offline/online decomposition and intra-session parallelism, whereas the semi-honest EMP backend used by *w/o Verif.* does not. A serial ablation of VeriANN (single session, online GC only) reaches 4.85 s on SIFT and 19.51 s on MS-MARCO, showing that the underlying verifiability overhead remains modest even without the parallelism advantage. Client–server communication is nearly identical.

Compared with *Strawman*, VeriANN is slower by 2.5× on SIFT and 1.7× on MS-MARCO because *Strawman* offloads post-processing to the client, while VeriANN executes the full top-k pipeline inside the GC to preserve dataset confidentiality. Even so, VeriANN reduces client–server communication by about 2.4× by avoiding per-bucket Merkle proofs to the client.

Compared with *Compass*, VeriANN remains in a comparable latency range while reducing client–server communication by three to four orders of magnitude. Unlike *Compass*, whose interactive client-driven ORAM traversal is highly network-sensitive, VeriANN requires only a single client–server round trip and is computation-bound. Moreover, VeriANN provides stronger security and deployment support: *Compass* assumes a single-client setting where the client holds the ORAM key, and thus offers no dataset confidentiality against the querying client.

Compared with *Perco*, VeriANN has comparable latency while reducing client–server communication by 3.8× on MS-MARCO and 2.2× on SIFT. We use the same (l, p, s) parameters for fairness, although *Perco*'s MS-MARCO accuracy remains low (56.8%) and scales poorly with more tables and probes. This gap is structural: *Perco*'s first-non-empty-bucket heuristic requires lookup tables of size at least 2^{35} to control spurious collisions. This also explains why extending *Perco* with verifiability is impractical: committing to tables of this size would be prohibitively expensive.

[2]The *w/o Verif.* baseline has no offline phase, so on a total-time-per-query basis it would have an advantage over VeriANN; here we focus on online latency, since VeriANN's offline phase is query-independent and can be amortized over a circuit pool (§8.2).

References

[1] Amazon Web Services. 2024. Amazon Bedrock: Foundation models and retrieval-augmented generation on AWS. https://aws.amazon.com/bedrock/. https://aws.amazon.com/bedrock/ Accessed: 2025-10-13.

[2] Amazon Web Services. 2026. AWS Interconnect: Managed Multicloud and Last-Mile Connectivity. https://docs.aws.amazon.com/interconnect/latest/userguide/what-is.html Direct AWS-to-Google Cloud interconnect with bandwidth up to 100 Gbps; standard hosted tiers include 1/2/5/10/25 Gbps. Accessed: 2026-04-30.

[3] Maya Anderson, Guy Amit, and Abigail Goldsteen. 2024. Is My Data in Your Retrieval Database? Membership Inference Attacks Against Retrieval Augmented Generation. arXiv:2405.20446

[4] Apple Inc. 2021. iCloud Private Relay Overview. https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf. Two-relay architecture; per Apple's whitepaper, user requests traverse two separate relays so that no single entity can combine identity and browsing activity. Accessed: 2026-04-30..

[5] Hilal Asi, Fabian Boemer, Nicholas Genise, Muhammad Haris Mughees, Tabitha Ogilvie, Rehan Rishi, Kunal Talwar, Karl Tarbe, Akshay Wadia, Ruiyu Zhu, and Marco Zuliani. 2024. Scalable Private Search with Wally. *arXiv preprint arXiv:2406.06761* (2024).

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function Secret Sharing. In *Advances in Cryptology – EUROCRYPT 2015 (Lecture Notes in Computer Science, Vol. 9057)*. Springer, 337–367.

[7] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 668–679.

[8] 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)*. 2111–2128.

[9] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. 2023. Authenticated private information retrieval. In *Proceedings of the 32nd USENIX Conference on Security Symposium* (Anaheim, CA, USA) *(SEC '23)*. USENIX Association, USA, Article 215, 17 pages.

[10] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 259–282.

[11] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2017. Riposte: An Anonymous Messaging System Handling Millions of Users. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 321–338.

[12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 643–662.

[13] Emma Dauterman, Vivian Fang, Natacha Crooks, and Raluca Ada Popa. 2022. Reflections on Trusting Distributed Trust. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks (HotNets)*.

[14] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. 2020. DORY: An Encrypted Search System with Distributed Trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 1101–1119.

[15] Daniel Demmler, T. Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *Network and Distributed System Security Symposium*.

[16] Gelei Deng, Yi Liu, Kailong Wang, Yuekang Li, Tianwei Zhang, and Yang Liu. 2024. PANDORA: Jailbreak GPTs by Retrieval Augmented Generation Poisoning.

[17] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 541–552.

[18] Xinyu Gao, Xiangtao Meng, Yingkai Dong, Zheng Li, and Shanqing Guo. 2025. DCMI: A Differential Calibration Membership Inference Attack Against Retrieval-Augmented Generation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25)*. 4184–4198.

[19] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint arXiv:2312.10997* (2024).

[20] Tim Geoghegan, Christopher Patton, Brandon Pitman, Eric Rescorla, and Christopher A. Wood. 2025. Distributed Aggregation Protocol for Privacy Preserving Measurement. IETF Internet-Draft draft-ietf-ppm-dap-17.

[21] Yuyang Gong, Zhuo Chen, Jiawei Liu, Miaokun Chen, Fengchang Yu, Wei Lu, XiaoFeng Wang, and Xiaozhong Liu. 2025. Topic-FlipRAG: Topic-Orientated Adversarial Opinion Manipulation Attacks to Retrieval-Augmented Generation Models. In *34th USENIX Security Symposium (USENIX Security 25)*. 3807–3826.

[22] Google Cloud. 2024. Google Cloud Vertex AI: Multimodal foundation models and RAG pipelines. https://cloud.google.com/vertex-ai. https://cloud.google.com/vertex-ai Accessed: 2025-10-13.

[23] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. 2023. Private Web Search with Tiptoe. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 396–416.

[24] Yan Huang, Jonathan Katz, and David Evans. 2013. Efficient secure two-party computation using symmetric cut-and-choose. In *Advances in Cryptology – CRYPTO 2013*.

[25] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 604–613.

[26] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2011).

[27] Yanze Jiang, Xinyang Yang, Xuanming Liu, Yanpei Guo, and Jiaheng Zhang. 2026. zkRAG: Efficiently Proving RAG Retrieval in Zero Knowledge. Cryptology ePrint Archive, Paper 2026/709.

[28] Daniel Kales. 2019. DPF-CPP: Distributed Point Function based PIR implementation in C++. https://github.com/dkales/dpf-cpp. Accessed: 2026-04-26.

[29] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. 2019. Revisiting User Privacy for Certificate Transparency. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[30] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 830–842.

[31] Minseok Kim, Hankook Lee, and Hyungjoon Koo. 2025. Rescuing the Unpoisoned: Efficient Defense against Knowledge Corruption Attacks on RAG Systems. In *Proceedings of the 2025 Annual Computer Security Applications Conference (ACSAC)*.

[32] Mehmet S. Kiraz and Berry Schoenmakers. 2006. A Protocol Issue for the Malicious Case of Yao's Garbled Circuit Construction. In *Proceedings of the Security in Information Technology and Biomedicine (SITB)*.

[33] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved Garbled Circuit: Free XOR Gates and Applications. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008), Part II (Lecture Notes in Computer Science, Vol. 5126)*. Springer, 486–498.

[34] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. 2012. Efficient similarity search over encrypted data. In *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 1156–1167.

[35] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2016. Riffle: An Efficient Communication System with Strong Anonymity. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2016, 2 (2016), 115–134.

[36] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.

[37] Jingyu Li, Zhicong Huang, Min Zhang, Cheng Hong, Jian Liu, Tao Wei, and Wenguang Chen. 2025. Panther: Private Approximate Nearest Neighbor Search in the Single Server Setting. 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, 365–379.

[38] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*. 950–961.

[39] Microsoft Azure. 2024. Azure OpenAI Service: Integrating large language models with enterprise data. https://learn.microsoft.com/en-us/azure/ai-services/openai/

[40] John X. Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M. Rush. 2023. Text Embeddings Reveal (Almost) As Much As Text. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[41] Ali Naseh, Yuefeng Peng, Anshuman Suri, Harsh Chaudhari, Alina Oprea, and Amir Houmansadr. 2025. Riddle Me This! Stealthy Membership Inference for Retrieval-Augmented Generation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25)*. 1245–1259.

[42] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated Machine Reading Comprehension Dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches 2016*.

[43] 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)*. IEEE, 911–929.

[44] Avital Shafran, Roei Schuster, and Vitaly Shmatikov. 2025. Machine Against the RAG: Jamming Retrieval-Augmented Generation with Blocker Documents. In *34th USENIX Security Symposium (USENIX Security 25)*. 3787–3806.

[45] Congzheng Song and Ananth Raghunathan. 2020. Information Leakage in Embedding Models. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer*

and Communications Security (CCS '20).
[46] Shang Song, Lin Liu, Rongmao Chen, Wei Peng, and Yi Wang. 2023. Secure Approximate Nearest Neighbor Search with Locality-Sensitive Hashing. In *European Symposium on Research in Computer Security*. Springer, 411–430.
[47] Xiao Wang, T.-H. Hubert Chan, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.
[48] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-sh2pc: Semi-honest Two-Party Computation based on Garbled Circuits. https://github.com/emp-toolkit/emp-sh2pc. Accessed: 2026-04-26.
[49] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. EMP-ag2pc: Authenticated Garbling and Efficient Malicious Two-Party Computation. https://github.com/emp-toolkit/emp-ag2pc. Accessed: 2026-04-26.
[50] Yuhao Wang, Wenjie Qu, Shengfang Zhai, Yanze Jiang, Zichen Liu, Yue Liu, Yinpeng Dong, and Jiaheng Zhang. 2025. Silent Leaks: Implicit Knowledge Extraction Attack on RAG Systems through Benign Queries. arXiv:2505.15420 [cs.CR]
[51] Chong Xiang, Tong Wu, Zexuan Zhong, David Wagner, Danqi Chen, and Prateek Mittal. 2026. Certifiably Robust RAG against Retrieval Corruption. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML) 2026*. Accepted papers list entry; final proceedings metadata (pages/DOI) pending..
[52] Lei Xu, Leqian Zheng, Chengzhi Xu, Xingliang Yuan, and Cong Wang. 2023. Leakage-abuse attacks against forward and backward private searchable symmetric encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 3003–3017.
[53] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, Chicago, Illinois, USA, 160–164.
[54] Kai Ye, Liangcai Su, and Chenxiong Qian. 2025. ImportSnare: Directed 'Code Manual' Hijacking in Retrieval-Augmented Code Generation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25)*. 335–349.
[55] Xingliang Yuan, Helei Cui, Xinyu Wang, and Cong Wang. 2015. Enabling privacy-assured similarity retrieval over millions of encrypted records. In *European symposium on research in computer security*. Springer, 40–60.
[56] Xingliang Yuan, Xinyu Wang, Cong Wang, Chenyun Yu, and Sarana Nutanong. 2017. Privacy-preserving similarity joins over encrypted data. *IEEE Transactions on Information Forensics and Security* 12, 11 (2017), 2763–2775.
[57] Shenglai Zeng, Jiankun Zhang, Pengfei He, Yue Xing, Yiding Liu, Han Xu, Jie Ren, Shuaiqiang Wang, Dawei Yin, Yi Chang, and Jiliang Tang. 2024. The Good and the Bad: Exploring Privacy Issues in Retrieval-Augmented Generation (RAG). arXiv:2402.16893
[58] Mingxun Zhou, Elaine Shi, and Giulia Fanti. 2025. Pacmann: Efficient Private Approximate Nearest Neighbor Search. In *The Thirteenth International Conference on Learning Representations (ICLR)*.
[59] Jinhao Zhu, Liana Patel, Matei Zaharia, and Raluca Ada Popa. 2025. Compass: encrypted semantic search with high accuracy. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. 915–938.
[60] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2025. PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models. In *34th USENIX Security Symposium (USENIX Security 25)*. 3827–3844.

A Experiment Environment

We implement VeriANN on top of EMP-toolkit [48, 49] for both semi-honest and authenticated garbling, and use the DPF construction of [29] (implementation [28]). The upstream EMP-toolkit runs single-threaded; we extend its authenticated garbling backend with gate-level parallelism via OpenMP, processing gates within the same topological layer of the circuit concurrently, since they share no data dependencies. The LSH index construction follows [43]. Each server is equipped with 8 cores of an Intel Xeon Platinum 8358 CPU @ 2.60 GHz and 512 GiB of memory, running Ubuntu 22.04 LTS.

B Notations in VeriANN

Table 5 summarizes the notation used throughout VeriANN.

Table 5: Notation Summary

Notation	Description
$\mathbf{x}$	Boldface symbols denote lists.
$\widetilde{x}$	Tilde notation denotes encrypted content.
$x^{(j)}$	Superscript (j) denotes ownership by Party j.
$[\![x]\!]$	XOR secret sharing of value x.
D	Embedding vector database.
$\mathbf{v}_i, \mathbf{v}_q$	Embedding of document d_i and query.
l	Total number of LSH hash tables in the index.
$\mathcal{H} = \{h_\ell\}_{\ell=0}^{l-1}$	LSH family.
T	LSH hash tables.
N_T	Number of buckets per hash table.
p	Number of probes per query.
B	A bucket containing identifiers.
s	Normalized (padded) bucket size.
k	Number of top results returned per query.
I	Outsourced searching index.
rt	Merkle root.
$\pi_{\ell,B}$	Merkle inclusion proof of bucket B.
st_D	Dataset state
st_Q	Per-query state:
λ	Security parameter (in bits)
$H(\cdot)$	Collision resistant hash function.
$\mathsf{AES}(x, sk, \eta)$	AES on payload x under key sk with nonce η.

C State-Pool Design

This section expands on the modular pre-computation design of §6.4, focusing on how the precomputed state pools in Figure 6 are organized and dispatched.

Pool structure. The expensive pre-selection computation—bucket decryption, Merkle-root reconstruction, probe consolidation, and cross-table merge-and-count—is identical for all $k \in \mathcal{K}$. Only the final extraction of k identifiers depends on k. We therefore precompute PP states for each index configuration (s, d, p, l) and separate TK_k states for each admissible k. More precisely, TK_k is indexed by the output size k and the input shape $w = p \cdot s$. At query time, the servers draw a fresh precomputed PP state from $\mathsf{Pool}_{\mathsf{PP}}$ for the matching configuration and a fresh TK_k state from $\mathsf{Pool}_{\mathsf{TK}}[k]$, corresponding to the lookups $st_{\mathsf{PP}} \leftarrow \mathsf{Pool}_{\mathsf{PP}}^{(j)}$ and $st_{\mathsf{TK}} \leftarrow \mathsf{Pool}_{\mathsf{TK}}^{(j)}[k]$ in Protocol 2. Each state is consumed once; the reusable object is the pool indexed by circuit shape, not a single garbled state.

Cross-dataset reuse. The TK_k pool can be shared across datasets whenever the PP output shape is identical: the same $w = p \cdot s$, identifier width, frequency width, root/tag length, and output encoding. Each PP circuit pads or truncates its consolidated output to this uniform size before committing. Dummy entries are assigned the lowest rank and are ignored by TK_k whenever at least k valid candidates are present; admissible values of k are restricted by the deployment set $\mathcal{K}$.

Online dispatch. Figure 6 illustrates the end-to-end flow. Offline, AuthGC.Init precomputes and buffers one-time states for PP and TK_k, indexed by the relevant configuration parameters. Online, each query consumes matching states from the two pools and feeds them into AuthGC.Online: the PIR outputs are first processed by

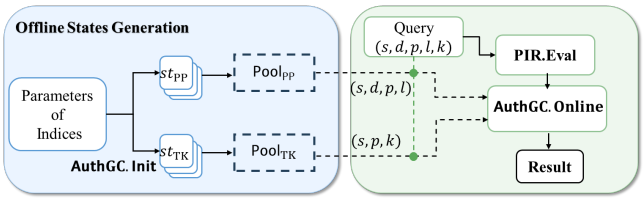


Figure 6: Modular circuit-pool design. Offline, AuthGC.Init precomputes and buffers garbled-circuit states for PP and TK indexed by index parameters. Online, each query dispatches matching states from each pool into AuthGC.Online, which pipelines the PIR outputs through PP and TK to produce the result.

PP, whose authenticated boundary output is then passed to the selected TK_k state to produce the final result.

D Additional Evaluations

D.1 Complexity Analysis

Table 6 summarizes the asymptotic complexity of each protocol phase. Below we highlight the dominant cost terms.

Query and Recover. Both client-side phases are lightweight and independent of the database size: the client only generates $l \cdot p$ DPF keys at query time and performs $O(k + l \cdot p)$ operations to reconstruct and verify the result.

PIR. This phase dominates server-side computation. Per (l, p) pair, each server performs a full-domain DPF evaluation ($O(N_T \log N_T)$ PRG calls) and aggregates row entries via XOR ($O(N_T \cdot (s + \log N_T))$).

GC-off. The offline GC cost scales linearly with the total number of AND gates (XOR gates are free under the free-XOR optimization [33]) across the two sub-circuits: the post-processing circuit PP (Circuit 1) and the top-k selection circuit TK (Circuit 2), which depends only on (k, w). Server-to-server communication scales with circuit size in the same way.

GC-on. The online computation mirrors the offline function-dependent phase, scaling as $O(|C|)$ symmetric-key operations per AND gate. The online communication, however, is substantially cheaper, scaling with circuit input/output size $O(|I| + |O|)$ rather than circuit size.

D.2 Post-Processing Design Ablation

We evaluate our ranking algorithm against two alternatives: a naive all-pairs oblivious frequency counter $O(n^2)$, and in-circuit Euclidean distance over d-dimensional embeddings $O(nd)$. Our hierarchical sort-based frequency counter runs in $O(n \log^2 w)$. Asymptotic costs are summarized in Table 7.

The frequency-based variants run at $n = p \cdot l \cdot s$ as fixed by each dataset's index configuration (Table 2), giving $n \in \{1024, 2048, 4096\}$ for SIFT, MS-MARCO-Sub, and MS-MARCO. For the distance variant we pick a modest $n \in \{32, 64, 128\}$ so its circuit fits on a comparable scale; moreover, the reported numbers cover only the distance-computation stage and *exclude* the subsequent oblivious sort, highlighting how expensive computing distances over high-dimensional embeddings is inside a garbled circuit.

Table 6: Asymptotic complexity per protocol phase. N_T: buckets per table; l: #tables per query; p: #probes per table; s: bucket size; d=$O(\log N_T)$: Merkle path length; w=p·s: candidate set size; k: output size; $|C_F|$: AND gate count of sub-circuit $F \in \{\mathsf{PP}(s, d, p, l), \mathsf{TK}(k, w)\}$. Communication direction: C→S (client-to-server), S↔S (server-to-server).

Phase	Computation	Comm. [Dir.]
Query (**C**)	$O(l \cdot p \log N_T)$	$O(l \cdot p \log N_T)$ **[C→S]**
PIR (**S**)	$O(l \cdot p \cdot N_T \log N_T)$ + $O(l \cdot p \cdot N_T \cdot (s + \log N_T))$	—
GC-Off (**S**)	$O(\lvert C_{\mathsf{PP}}\rvert + \lvert C_{\mathsf{TK}}\rvert)$	$O(\lvert C_{\mathsf{PP}}\rvert + \lvert C_{\mathsf{TK}}\rvert)$ **[S↔S]**
GC-On (**S**)	$O(\lvert C_{\mathsf{PP}}\rvert + \lvert C_{\mathsf{TK}}\rvert)$	$O(l \cdot p \cdot (s + \log N_T) + w + k)$ **[S↔S]**
Recover (**C**)	$O(k + p \cdot l)$	$O(k)$ **[S→C]**

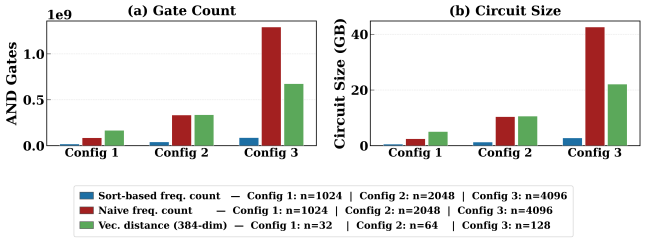


Figure 7: Ranking algorithm comparison across three index configurations. Frequency-based variants use $n = p \cdot \ell \cdot s$ from Table 2; the distance variant uses a 32× smaller n to fit on the same scale, and reports only the distance-computation stage (oblivious sort excluded).

Table 7: Computational complexity of ranking algorithms. n: candidate-set size; d: embedding dimension; $w = p \cdot s$: per-table candidate width.

Naive freq. count	Vec. distance	Sort-based (ours)
$O(n^2)$	$O(nd)$	$O(n \log^2 w)$

Figure 7 reports AND-gate count and Bristol-format circuit size. At the largest setting, the naive counter is 14.6× larger than ours (1.29G vs. 88.6M gates). Even at this modest n and without counting its sort stage, the distance variant is still 7.6× larger (677M gates, 22.2 GB), and additionally inflates the index by a factor of d, since each bucket entry must store the full embedding rather than a single identifier. The gap also widens with scale: as n grows from 1024 to 4096, our circuit grows by only 4.4× while the naive counter grows by 14.9×, matching the asymptotic separation between $O(n \log^2 w)$ and $O(n^2)$.

D.3 Retrieval Accuracy

We next examine how our ranking design affects retrieval quality. We follow the same c-ANN accuracy metric and setting as [43, 46]:

the fraction of returned results that are c-ANNs of the corresponding query vectors. Figure 8 reports accuracy on SIFT, MS-MARCO-Sub, and MS-MARCO as we sweep the number of hash tables $l \in \{4, 8, 16, 32\}$, the number of probes per table $p \in \{1, 2, 4, 8\}$, and the bucket size $s \in \{32, 64, 128\}$, comparing plaintext direct frequency counting (DirectCount) against our oblivious hierarchical sort-based counter (HierSort).

Frequency counting scales gracefully with index parameters. Across all three datasets, DirectCount exhibits the expected monotone behavior: accuracy rises consistently with l, p, and s, and saturates near the 90% line at moderate settings (e.g., l=16, $p \geq 4$). This confirms that frequency-based ranking is a faithful proxy for embedding similarity at our target operating points, and that retrieval quality can be tuned smoothly by adjusting any of the three parameters rather than relying on a single dominant knob.

Hierarchical sorting preserves the scaling trend within one parameter step. HierSort tracks DirectCount closely throughout the sweep, preserving the same monotone scaling in l, p, and s. Where a small gap appears, it is consistently bounded by a single parameter step: the accuracy reached by HierSort at a given configuration matches that of DirectCount after a one-notch increase in L, p, or s (e.g., on MS-MARCO at s=64, HierSort with L=16 matches DirectCount with L=8). Intuitively, our hierarchical aggregation counts how many *tables* an identifier appears in rather than how many *probes*, which slightly compresses the frequency dynamic range; this is easily compensated by a marginal bump in any one index parameter.

This accuracy gap is a worthwhile trade for the resulting circuit savings. Recall from Section D.2 that hierarchical sorting reduces the post-processing circuit from $O(n^2)$ to $O(n \log^2 w)$, yielding a substantial reduction in AND gates and circuit storage at large settings. Recovering the residual accuracy gap costs at most one step in L, p, or s, whereas the savings on the GC side are an order of magnitude. We therefore consider the accuracy cost of hierarchical sorting to be fully acceptable in the authenticated-garbling regime targeted by VeriANN.

D.4 Modular design supports multiple top-k values at negligible extra cost.

Figure 9 reports the total circuit pool size as $|\mathcal{K}|$ grows from 1 to 5. The non-modular baseline must garble and store a full pipeline circuit for every $k \in \mathcal{K}$, so both the precomputation cost and state pool size grow linearly with $|\mathcal{K}|$. Our modular design instead precomputes a single shared PP circuit plus one lightweight TK_k per supported k; since TK accounts for less than 1.3% of the full circuit, the total pool size remains essentially flat as $|\mathcal{K}|$ increases. Notably, at $|\mathcal{K}| = 1$ both designs incur nearly identical cost, confirming that modularization itself introduces negligible overhead and the savings grow proportionally with the number of supported top-k values.

E Selective Failure via Inconsistent Rows.

For simplicity, assume each server stores four data rows $\{x_0, x_1, x_2, x_3\}$, where each row x_i is appended with a corresponding Merkle proof π_i attesting its inclusion under a public Merkle root.

Suppose a malicious server ($\mathcal{S}_1$) suspects that the client is likely to query the second entry x_2. To launch a selective failure attack,

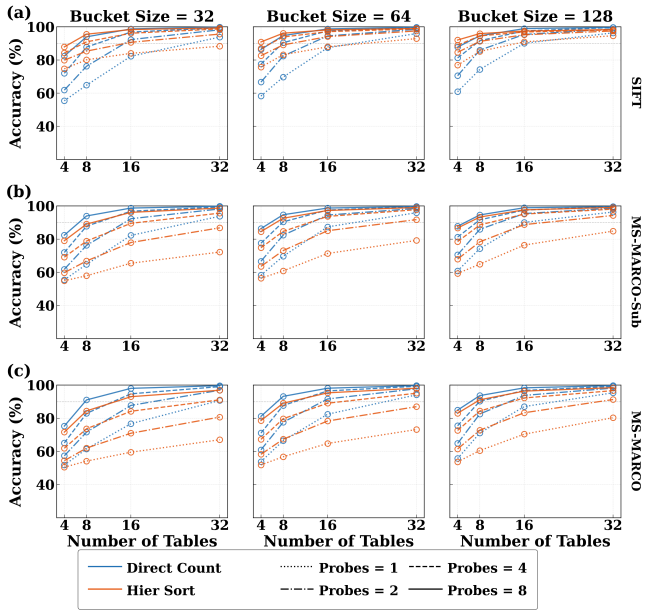


Figure 8: Accuracy under varying number of tables L, probes per table p, and bucket size s on (a) SIFT, (b) MS-MARCO-Sub, and (c) MS-MARCO. DirectCount (blue) is the plaintext frequency counting; HierSort (orange) is our oblivious hierarchical sort-based variant. The two curves track each other within one parameter step across all configurations.

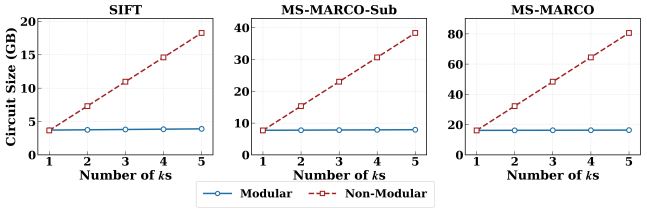


Figure 9: Circuit size in Bristol format under varying $|\mathcal{K}|$ for the modular and non-modular designs across three datasets. circuit size directly reflects the state pool size.

$\mathcal{S}_1$ replaces the second row $(x_2 \,\|\, \pi_2)$ with another valid row, for example $(x_1 \,\|\, \pi_1)$. As a result, the databases stored by the two servers become:

$$\text{Server } \mathcal{S}_0\text{: } \{x_0 \,\|\, \pi_0,\ x_1 \,\|\, \pi_1,\ x_2 \,\|\, \pi_2,\ x_3 \,\|\, \pi_3\},$$

$$\text{Server } \mathcal{S}_1\text{: } \{x_0 \,\|\, \pi_0,\ x_1 \,\|\, \pi_1,\ x_1 \,\|\, \pi_1,\ x_3 \,\|\, \pi_3\}.$$

Case 1: Querying a non-target row. Assume the client queries an index other than 2, e.g., row 0. Let the local evaluation vectors at the two servers be $\mathbf{v}_0 = (0, 1, 1, 0)$ and $\mathbf{v}_1 = (1, 1, 1, 0)$, respectively. The local aggregations computed by the servers are then:

$$\text{Server } \mathcal{S}_0\text{: } x_1 \,\|\, \pi_1 \ \oplus\ x_2 \,\|\, \pi_2,$$

$$\text{Server } \mathcal{S}_1\text{: } x_0 \,\|\, \pi_0 \ \oplus\ x_1 \,\|\, \pi_1 \ \oplus\ x_1 \,\|\, \pi_1.$$

Circuit 1 PostProcess (s, d, p, l) [PP]

Circuit parameters:
s: Bucket size; d: Merkle path length;
p: Probes per table; l: Number of tables; $w = p \cdot s$.
Input to Party$_j$, $j \in \{0,1\}$**:**
$\{\mathbf{a}_{m,n}^{(j)},\ [\![sk_m]\!]^{(j)}\},\ [\![st_Q]\!]^{(j)}$

Phase I: Per-bucket decryption and Merkle root reconstruction
$\{b_{m,n}\}_{m,n}, r \leftarrow [\![st_Q]\!]^{(0)} \oplus [\![st_Q]\!]^{(1)}$
for $m \in [l],\ n \in [p]$ **do**
  $(\mathbf{id}_{m,n},\ rt_{m,n}) \leftarrow$ DecryptReconstruct$(\mathbf{a}_{m,n},\ sk_m,\ b_{m,n})$
end for
Phase II: Per-table probe consolidation
for $m \leftarrow 0$ to $l-1$ **do**
  $b_eq \leftarrow$ AssertEq$(rt_{m,0}, \ldots, rt_{m,p-1})$
  $rt_m^{\text{tbl}} \leftarrow$ Select $(b_eq, rt_{m,0}, rt_{\text{dum}})$
  $(\mathbf{id}_m,\ \mathbf{freq}_m) \leftarrow$ Consolidate$(\{\mathbf{id}_{m,n}\}_{n\in[p]})$
end for
Phase III: Hierarchical cross-table merge-and-count
for $h \leftarrow 1$ to $\log_2 l$ **do**
  for $i \leftarrow 0$ to $l/2^h - 1$ **do**
    $(\mathbf{id}, \mathbf{freq}) \leftarrow$ MergeCount(level-$(h-1)$ children at i)
  end for
end for
$rt^{\text{tbl}} \leftarrow H(rt_1^{\text{tbl}} \| \ldots \| rt_l^{\text{tbl}})$
Phase IV: Output binding to TK
$tag_q \leftarrow H(\{b_{m,n}\}_{m,n} \| r)$
$rt' \leftarrow rt^{\text{tbl}} \oplus H(\mathbf{id} \,\|\, \mathbf{freq} \,\|\, tag_q)$
return $(rt',\ \mathbf{id},\ \mathbf{freq},\ tag_q)$

After reconstruction, the client obtains

$$(x_1 \oplus x_2) \,\|\, (\pi_0 \oplus \pi_1),$$

where $(\pi_0 \oplus \pi_1)$ does not constitute a valid Merkle proof for $(x_1 \oplus x_2)$. Consequently, verification fails and the client rejects the result.

Case 2: Querying the targeted row. Now assume the client queries entry x_2 and the local evaluation bit corresponding to row 2 at the malicious server ($\mathcal{S}_1$) is 1 (which occurs with non-negligible probability), the local evaluation vectors become $\mathbf{v}_0 = (0,1,0,1)$ and $\mathbf{v}_1 = (0,1,1,1)$. The corresponding local aggregations are:

$$\text{Server } \mathcal{S}_0\text{: } x_1 \,\|\, \pi_1 \ \oplus\ x_3 \,\|\, \pi_3.$$

$$\text{Server } \mathcal{S}_1\text{: } x_1 \,\|\, \pi_1 \ \oplus\ x_1 \,\|\, \pi_1 \ \oplus\ x_3 \,\|\, \pi_3.$$

After reconstruction, the client obtains

$$x_1 \,\|\, \pi_1,$$

where π_1 is a valid Merkle proof for x_1. Thus, the client accepts the result.

Remark. In standard authenticated PIR [9], this attack is trivially defeated by embedding the row index i within each data row: after reconstruction, the client simply checks whether the recovered index matches its query target, and aborts if not. In VeriANN, however, bucket contents are never revealed to the client, so the client has no opportunity to inspect individual row identities mid-protocol. This motivates our construction in §6.1, which addresses this selective failure attack.

Circuit 2 SelectTopK (k, w) [TK]

Circuit parameters:
k: Number of returned items; w: Candidate set size.
Input to Party$_j$, $j \in \{0,1\}$**:**
$[\![\mathbf{id}]\!]^{(j)},\ [\![\mathbf{freq}]\!]^{(j)},\ [\![rt']\!]^{(j)},\ [\![tag_q]\!]^{(j)}$

Phase I: State reconstruction
$\mathbf{id} \leftarrow [\![\mathbf{id}]\!]^{(0)} \oplus [\![\mathbf{id}]\!]^{(1)}$
$\mathbf{freq} \leftarrow [\![\mathbf{freq}]\!]^{(0)} \oplus [\![\mathbf{freq}]\!]^{(1)}$
$tag_q \leftarrow [\![tag_q]\!]^{(0)} \oplus [\![tag_q]\!]^{(1)}$
$rt \leftarrow [\![rt']\!]^{(0)} \oplus [\![rt']\!]^{(1)} \oplus H(\mathbf{id} \,\|\, \mathbf{freq} \,\|\, tag_q)$
Phase II: Top-k selection
BitonicSort$^*(\mathbf{freq},\ \mathbf{id},\ w, k)$
// sort by frequency in descending order
$\mathbf{id}_{\text{top-}k} \leftarrow \mathbf{id}[0{:}k]$
Phase III: Commitment generation
$cm \leftarrow H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, rt)$
// rt equals the public index root rt_I under honest execution
return $(\mathbf{id}_{\text{top-}k},\ cm)$

Algorithm 1 DecryptReconstruct$(\mathbf{a},\ sk,\ b)$

Parse $\mathbf{a}$ as $(\tilde{\mathbf{B}},\ \mathbf{path},\ \mathbf{dirs})$
$|\tilde{\mathbf{B}}| = s \cdot 32$ bits, $|\mathbf{path}| = 256d$ bits, $|\mathbf{dirs}| = d$ bits
$\mathbf{id} \leftarrow \text{AES}(\tilde{\mathbf{B}},\ sk,\ b)$
$leaf \leftarrow H(b \,\|\, \tilde{\mathbf{B}} \,\|\, sk)$
for $j \leftarrow 1$ to d **do**
  $L \leftarrow$ Select$(\mathbf{dirs}[j],\ \mathbf{path}[j],\ leaf)$
  $R \leftarrow$ Select$(\mathbf{dirs}[j],\ leaf,\ \mathbf{path}[j])$
  $leaf \leftarrow H(L \,\|\, R)$
end for
return $(\mathbf{id},\ leaf)$

Algorithm 2 Consolidate$(\{\mathbf{id}_n\}_{n\in[p]})$

$\mathbf{id} \leftarrow \mathbf{id}_0 \,\|\, \mathbf{id}_1^{\leftarrow} \,\|\, \mathbf{id}_2 \,\|\, \mathbf{id}_3^{\leftarrow} \,\|\, \cdots$
// even-indexed in original order, odd-indexed reversed
BitonicSort$^*(\mathbf{id},\ p,\ s)$
for $j \leftarrow 0$ to $p-1$ **do**
  // boundary: compare against the j predecessors that exist
  $b_{\text{dup}} \leftarrow \bigvee_{t=1}^{j}$ Equal$(\mathbf{id}[j],\ \mathbf{id}[j-t])$
  $\mathbf{freq}[j] \leftarrow \neg b_{\text{dup}}$
  $\mathbf{id}[j] \leftarrow$ Select$(b_{\text{dup}},\ \perp,\ \mathbf{id}[j])$
end for
for $j \leftarrow p$ to $p \cdot s - 1$ **do**
  $b_{\text{dup}} \leftarrow \bigvee_{t=1}^{p-1}$ Equal$(\mathbf{id}[j],\ \mathbf{id}[j-t])$
  $\mathbf{freq}[j] \leftarrow \neg b_{\text{dup}}$
  $\mathbf{id}[j] \leftarrow$ Select$(b_{\text{dup}},\ \perp,\ \mathbf{id}[j])$
end for
return $(\mathbf{id},\ \mathbf{freq})$

E.1 Subroutine

Circuits PP and TK together implement the secure oblivious top-k retrieval pipeline. PP (Circuit 1) takes the raw PIR aggregation outputs and produces a deduplicated, frequency-annotated candidate list along with a blinded Merkle root for integrity binding. TK (Circuit 2) consumes the output of PP, selects the k most frequent candidates, and emits a verification commitment that the client uses to authenticate the result.

Algorithm 3 MergeCount$((\mathbf{id}_1, \mathbf{freq}_1), (\mathbf{id}_2, \mathbf{freq}_2))$

$\mathbf{id} \leftarrow \mathbf{id}_1 \parallel \mathbf{id}_2$; $\mathbf{freq} \leftarrow \mathbf{freq}_1 \parallel \mathbf{freq}_2$
BitonicSort$(\mathbf{id}, \mathbf{freq}, 2w)$
$\mathbf{freq}[0] \leftarrow \neg$IfDummy$(\mathbf{id}[0]) \cdot \mathbf{freq}[0]$
for $j \leftarrow 1$ to $2w-1$ **do**
  $b_{\mathsf{eq}} \leftarrow$ Equal$(\mathbf{id}[j], \mathbf{id}[j-1])$
  $b_{\mathsf{dum}} \leftarrow$ IfDummy$(\mathbf{id}[j])$
  $\mathbf{freq}[j-1] \leftarrow \mathbf{freq}[j-1] + (b_{\mathsf{eq}} \wedge \neg b_{\mathsf{dum}}) \cdot \mathbf{freq}[j]$
  $\mathbf{freq}[j] \leftarrow \mathbf{freq}[j] \cdot (b_{\mathsf{eq}} \vee b_{\mathsf{dum}})$
end for
BitonicSort$^*(\mathbf{freq}, \mathbf{id}, 2w)$
return $(\mathbf{id}[0{:}w], \mathbf{freq}[0{:}w])$

Circuit PP. The circuit proceeds in four phases. Phase I invokes DecryptReconstruct (Algorithm 1) for each retrieved bucket, which decrypts the ciphertext via AES and reconstructs the per-bucket Merkle root by hashing the leaf $H(b\|\mathbf{id}\|sk)$ up the sibling path. Phase II calls Consolidate (Algorithm 2) per table, which sorts the p probe lists obliviously and performs a linear scan to deduplicate identifiers, replacing duplicates with dummy entries $\perp$ and assigning each distinct identifier a frequency of 1. Phase III repeatedly applies MergeCount (Algorithm 3) in a $\log_2 l$-round hierarchical reduction: each call sorts the concatenated pair of candidate lists by identifier, accumulates frequencies for matching entries, then re-sorts by frequency retaining only the top-w candidates. The second sort in both Consolidate and MergeCount uses BitonicSort*, which denotes the tailored bitonic sort optimizations described in §6.3. Phase IV binds the consolidated output to the query state by computing $tag_q \leftarrow H(\{b_{m,n}\}\|st_Q)$ and masking the Merkle root as $rt' \leftarrow rt^{\mathsf{tbl}} \oplus H(\mathbf{id}\|\mathbf{freq}\|tag_q)$.

Circuit TK. The circuit reconstructs the candidate list and unblinded index root from the shared inputs, invokes BitonicSort* to select the top-k identifiers by frequency—terminating early as described in §6.3—and outputs the commitment $cm \leftarrow H(\mathbf{id}_{\text{top-}k}\|tag_q\|rt)$ for client verification.

F Security Analysis

F.1 Verifiability: Formal Treatment and Proofs

This appendix gives the formal game underlying §7.1 and the proofs of all stated lemmas and Theorem 7.1. Throughout, $H : \{0,1\}^* \to \{0,1\}^\lambda$ is modelled as a random oracle, q_H bounds the adversary's H-queries, and ρ is AuthGC's statistical security parameter.

F.1.1 Refined protocol syntax. We refine the four-stage description of §6.5 into Π = (Build, Query, Answer, Transfer, Recover), where Build and Query are unchanged (in particular Query samples $r \xleftarrow{\$} \{0,1\}^\lambda$ and shares $st_Q = (r, \mathbf{b})$); Answer emits the PIR output $\mathbf{a}^{(j)}$; Transfer runs the PP AuthGC online phase and emits the boundary share $[\![\mathbf{y}_{\mathsf{PP}}]\!]^{(j)}$ encoding $(\mathbf{id}, \mathbf{freq}, rt', tag_q)$; and Recover runs the TK AuthGC instance and performs the final commitment check. Let $\Phi(\mathbf{v}_q, D, k) \in \mathsf{Ids}^k$ be the prescribed top-k output in the clear, defined relative to the committed index I produced by $\mathsf{Build}(1^\lambda, D)$ in step 1 of the experiment; Build's internal randomness (LSH hash family, AES keys, and bucket padding) is absorbed into the experiment's random tape, so Φ is deterministic given a fixed I.

F.1.2 Verifiability game. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a PPT two-phase adversary statically corrupting one server $b \in \{0,1\}$. Define $\mathsf{Exp}^{\mathsf{verif}}_{\Pi,\mathcal{A},b}(\lambda, D, \mathbf{v}_q, k)$ as follows.

(1) $(I, \mathsf{st}_D, rt_I) \leftarrow \mathsf{Build}(1^\lambda, D)$.
(2) $(q^{(0)}, q^{(1)}, [\![st_Q]\!]^{(0)}, [\![st_Q]\!]^{(1)}, st_C) \leftarrow \mathsf{Query}(1^\lambda, \mathbf{v}_q, k)$.
(3) $a^{(1-b)} \leftarrow Answer(1-b, I, q^{(1-b)}, [\![st_Q]\!]^{(1-b)})$.
(4) $(\mathbf{a}^{*(b)}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_0(I, rt_I, q^{(b)}, [\![st_Q]\!]^{(b)}, [\![\{sk_{T,m}\}]\!]^{(b)})$.
(5) $([\![\mathbf{y}_{\mathsf{PP}}]\!]^{(b)}, [\![\mathbf{y}_{\mathsf{PP}}]\!]^{(1-b)}) \leftarrow \mathsf{Transfer}(\mathbf{a}^{*(b)}, \mathbf{a}^{(1-b)}, [\![st_Q]\!]^{(b)}, [\![st_Q]\!]^{(1-b)}, [\![\{sk_{T,m}\}]\!]^{(b)}, [\![\{sk_{T,m}\}]\!]^{(1-b)})$.
(6) $[\![\mathbf{y}_{\mathsf{PP}}]\!]^{*(b)} \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, [\![\mathbf{y}_{\mathsf{PP}}]\!]^{(b)})$.
(7) $y \leftarrow \mathsf{Recover}([\![\mathbf{y}_{\mathsf{PP}}]\!]^{*(b)}, [\![\mathbf{y}_{\mathsf{PP}}]\!]^{(1-b)}, st_C)$; output 1 iff $y \notin \{\Phi(\mathbf{v}_q, D, k), \perp\}$.

where $\Phi(\mathbf{v}_q, D, k)$ denotes the deterministic cleartext execution of the VeriANN retrieval pipeline on the committed index I from step 1; equivalently, $\Phi(\mathbf{v}_q, D, k) = \mathsf{Top}_k(\mathbf{v}_q, \Pi_{\mathsf{ANN}}(\mathbf{v}_q, I))$ where Π_{ANN} is the prescribed LSH multi-probe + frequency aggregation pipeline. The notation hides the dependence on the build-time randomness for brevity.

The experiment exposes two adversarial choice points: $\mathcal{A}_0$ chooses $\mathbf{a}^{*(b)}$ (input-boundary tampering, §6.1) before the AuthGC PP execution, and $\mathcal{A}_1$ chooses $[\![\mathbf{y}_{\mathsf{PP}}]\!]^{*(b)}$ (cross-circuit boundary tampering, §6.3) after observing the prescribed boundary share $[\![\mathbf{y}_{\mathsf{PP}}]\!]^{(b)}$ that the AuthGC protocol would yield to the corrupted side. This models a malicious server that adapts its boundary tampering to the actual AuthGC output, rather than committing to it blindly.

Definition F.1 (Verifiability of VeriANN, formal). Π *satisfies* verifiability *with error* $\varepsilon(\lambda)$ *if for every PPT two-phase adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, every* $b \in \{0,1\}$*, and every* $(D, \mathbf{v}_q, k)$, $\Pr[\mathsf{Exp}^{\mathsf{verif}}_{\Pi,\mathcal{A},b} = 1] \le \varepsilon(\lambda)$*. We say* Π *is* verifiable *if* ε *is negligible.*

F.1.3 Supporting lemmas.

Lemma F.1 (Authenticated leaf binding). *For the leaf hash* $\ell_{m,i} = H(b_{m,i} \parallel \widetilde{B}_{m,i} \parallel sk_{T,m})$*, no PPT* $\mathcal{A}^H$ *produces colliding triples* $(b, \widetilde{B}, sk) \neq (b', \widetilde{B}', sk')$ *except with probability at most* $\binom{q_H}{2} \cdot 2^{-\lambda}$.

Proof. Birthday bound on the random oracle: each pair of distinct queries collides with probability $2^{-\lambda}$, and there are at most $\binom{q_H}{2}$ such pairs. □

Lemma F.2 (Per-query tag entropy). *For* $st_Q = (r, \mathbf{b})$ *with* $r \xleftarrow{\$} \{0,1\}^\lambda$ *and* $tag_q = H(\mathbf{b} \parallel r)$,

$$H_\infty(tag_q \mid \mathsf{view}_{\mathcal{A}}) \;\ge\; \lambda - \log q_H,$$

and $\Pr[\mathcal{A}$ *outputs the correct* $tag_q] \le q_H \cdot 2^{-\lambda}$.

Proof. By perfect hiding of XOR sharing [15], the corrupted server's share carries no information about st_Q, so conditioned on $\mathsf{view}_{\mathcal{A}} = (I, rt_I, q^{(b)}, [\![st_Q]\!]^{(b)})$, the value r is uniform in $\{0,1\}^\lambda$. Since H is a random oracle and r appears in its input, $H(\mathbf{b} \parallel r)$ is uniform conditioned on $\mathsf{view}_{\mathcal{A}}$ unless $\mathcal{A}$ queries H at the correct input, which by union bound happens with probability at most $q_H \cdot 2^{-\lambda}$. □

Lemma F.3 (Authenticated state-transfer). *Let* $tag_q = H(\mathbf{b} \,\|\, r)$ *and* $rt' = rt \oplus H(\mathbf{id} \,\|\, \mathbf{freq} \,\|\, tag_q)$. *Consider any PPT adversary* $\mathcal{A}$ *corrupting one server and given only its boundary share of* $(\mathbf{id}, \mathbf{freq}, rt', tag_q)$. *Suppose* $\mathcal{A}$ *replaces this share so that the reconstructed input to* TK *is* $(\mathbf{id}^*, \mathbf{freq}^*, rt'^*, tag_q^*)$ *with* $(\mathbf{id}^*, \mathbf{freq}^*) \neq (\mathbf{id}, \mathbf{freq})$. *Then, except with probability at most* $(q_H + 1) \cdot 2^{-\lambda}$, TK *recovers a root* $rt^* \neq rt$.

Proof. Let $\mu = H(\mathbf{id} \,\|\, \mathbf{freq} \,\|\, tag_q)$, $\mu^* = H(\mathbf{id}^* \,\|\, \mathbf{freq}^* \,\|\, tag_q^*)$. The event $rt^* = rt$ is equivalent to$\mu^* \oplus \mu = rt'^* \oplus rt'$, where the right-hand side is fixed by the adversary's tampered boundary share. The value $tag_q = H(\mathbf{b} \,\|\, r)$ is hidden from any single server by Lemma F.2, so $\mu = H(\mathbf{id} \,\|\, \mathbf{freq} \,\|\, tag_q)$ is uniform from $\mathcal{A}$'s perspective unless $\mathcal{A}$ queries H at the correct hidden input. The value $\mu^* = H(\mathbf{id}^* \,\|\, \mathbf{freq}^* \,\|\, tag_q^*)$ may be known to $\mathcal{A}$ (the inputs are all adversary-chosen), but since $(\mathbf{id}^*, \mathbf{freq}^*) \neq (\mathbf{id}, \mathbf{freq})$ the two oracle inputs are distinct whether or not $tag_q^* = tag_q$, so the values are sampled independently in the random-oracle model. The event $\mu^* \oplus \mu = rt'^* \oplus rt'$ then forces μ to equal a specific target $\mu^* \oplus rt'^* \oplus rt'$, which $\mathcal{A}$ can hit only by querying H at the correct hidden input. Thus the above equation holds with probability at most $2^{-\lambda}$ when $\mathcal{A}$ does not query the random oracle at an input containing the correct hidden tag_q, and with probability at most $q_H \cdot 2^{-\lambda}$ otherwise (Lemma F.2). Therefore the state transfer fails to preserve rt except with probability $(q_H + 1) \cdot 2^{-\lambda}$. □

The proof is agnostic to whether PP and TK share a single garbled circuit—only the unpredictability of tag_q matters—which is what makes the modular composition of §6.3 compatible with verifiability.

Proof of Theorem 7.1. Let y denote the client's output. We bound $\Pr[y \notin \{\Phi(\mathbf{v}_q, D, k), \perp\}]$ by union-bounding over the following events, conditioned in sequence.

E_{coll} *(leaf collision):* by Lemma F.1, $\Pr[E_{\mathsf{coll}}] \le \binom{q_H}{2} \cdot 2^{-\lambda}$. Outside this event, every leaf is uniquely identified by $(b, \widetilde{B}, sk_{T,m})$, so any input-boundary substitution either (i) triggers PP's in-circuit AssertEq across the p per-table probes to fail and substitutes a dummy root, or (ii) yields a per-table reconstructed root $rt_m^{\mathrm{tbl}} \neq rt_m$. In either case the aggregated root $rt^{\mathrm{tbl}} = H(rt_1^{\mathrm{tbl}} \| \cdots \| rt_l^{\mathrm{tbl}})$ propagated through the PP–TK boundary differs from the true rt_I, and the discrepancy ultimately surfaces at the client's *cm* check (event E_{cm} below).

E_{PP} *(PP in-circuit deviation):* the PP AuthGC instance deviates from its prescribed circuit while the honest party does not abort, with probability at most $2^{-\rho}$ by AuthGC binding.

E_{tx} *(cross-circuit tampering):* outside $E_{\mathsf{coll}} \cup E_{\mathsf{PP}}$, the PP output is the prescribed boundary value. Any boundary replacement that changes $(\mathbf{id}, \mathbf{freq})$ while causing TK to recover the same root succeeds with probability at most $(q_H + 1) \cdot 2^{-\lambda}$ by Lemma F.3.

E_{TK} *(TK in-circuit deviation):* after the boundary input is fixed, the TK AuthGC instance either evaluates the prescribed TK circuit on its received input or the honest party aborts, except with probability at most $2^{-\rho}$ by AuthGC binding.

E_{cm} *(commitment forgery):* conditioned on the complement of all the above, either the recovered root inside TK equals rt_I and the emitted commitment is $cm = H(\Phi(\mathbf{v}_q, D, k) \,\|\, tag_q \,\|\, rt_I)$, or the client aborts. Thus the client's check accepts a non-honest top-k only if $\mathcal{A}$ produces a second preimage for H, which by union bound happens with probability at most $q_H \cdot 2^{-\lambda}$.

Combining,

$$\varepsilon(\lambda) \le \binom{q_H}{2} \cdot 2^{-\lambda} + (2q_H + 1) \cdot 2^{-\lambda} + 2 \cdot 2^{-\rho},$$

which is negligible. □

F.2 Query Privacy: Formal Treatment and Proofs

This appendix gives the formal real-ideal game underlying §7.2 and the hybrid proof of Theorem 7.2, with the accept-or-abort bit included in the adversary's view.

F.2.1 Real-ideal game. Let $\Pi = (\mathsf{Build}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$ denote the VeriANN protocol. Fix a security parameter λ, dataset D, honest query embedding $\mathbf{v}_q$, admissible output size $k \in \mathcal{K}$, corrupted server $b \in \{0, 1\}$, auxiliary input z, and a two-phase adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$.

Real distribution. $\mathsf{REAL}_{\mathcal{A},b,D,\mathbf{v}_q,k,\lambda}(z)$ is sampled by running $\mathsf{Build}(1^\lambda, D) \to (I, st_D, \mathcal{K})$ and $\mathsf{Query}(1^\lambda, \mathbf{v}_q, k) \to (q^{(0)}, q^{(1)}, [\![st_Q]\!]^{(0)}, [\![st_Q]\!]^{(1)}, st_C)$, then invoking $\mathcal{A}_0(z, I, st_D, k, q^{(b)}, [\![st_Q]\!]^{(b)}) \to (\mathrm{st}_{\mathcal{A}}, \mathsf{cmd})$, executing $\mathsf{Exec}_b^{\mathcal{A}} \to (\tau^{(b)}, y)$, setting $\beta \leftarrow 1[y \neq \perp]$, and returning $\hat{\beta} \leftarrow \mathcal{A}_1(\mathrm{st}_{\mathcal{A}}, \tau^{(b)}, \beta)$. The transcript $\tau^{(b)}$ contains the corrupted server's PIR messages, AuthGC transcripts, output shares, and local randomness; cmd abstracts any adversarial deviation (malformed shares, boundary-state replacement, output-share tampering, etc.).

Ideal distribution. $\mathsf{IDEAL}_{\mathcal{A},\mathsf{Sim},D,k,\lambda}(z)$ is defined analogously, except that $\mathsf{Sim}_0(1^\lambda, I, st_D, k, z)$ produces $(\tilde{q}^{(b)}, \widetilde{[\![st_Q]\!]^{(b)}})$ in place of the real query, and $\mathsf{Sim}_1(\mathrm{st}_{\mathsf{Sim}}, \mathsf{cmd})$ produces $(\tilde{\tau}^{(b)}, \tilde{\beta})$ in place of the real transcript and accept bit. The simulator receives no information about $\mathbf{v}_q$; $\mathcal{A}$ and z are unchanged across both worlds.

Definition F.2 (Query Privacy, formal). Π *satisfies* query privacy *if for every* $b \in \{0, 1\}$ *and every PPT adversary* $\mathcal{A}$, *there exists a PPT simulator* Sim *such that for every* D, $\mathbf{v}_q$, *admissible* $k \in \mathcal{K}$, *and auxiliary input* z,

$$\mathsf{REAL}_{\mathcal{A},b,D,\mathbf{v}_q,k,\lambda}(z) \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{A},\mathsf{Sim},D,k,\lambda}(z).$$

F.2.2 Syntactic effective tampering.

Definition F.3 (Syntactic effective tampering). $\mathsf{EffTamper}(\mathsf{cmd}) = 1$ *if* cmd *contains a non-zero offset* $\Delta \neq 0$ *at one of: (i) the corrupted server's input to* PP *(PIR aggregation share or authentication data); (ii) its boundary share forwarded from* PP *to* TK*; (iii) its final output share to the client. Otherwise,* $\mathsf{EffTamper}(\mathsf{cmd}) = 0$. *The predicate is a polynomial-time deterministic function of* cmd, *independent of* $\mathbf{v}_q$ *and any honest-party secret. Explicit refusal to participate is treated as denial of service and is outside the query-privacy leakage channel.*

F.2.3 Supporting lemma.

Lemma F.4 (Abort bit under syntactic effective tampering). *Excluding denial-of-service behaviour, for every* λ, D, $\mathbf{v}_q$, *admissible* $k \in \mathcal{K}$, $b \in \{0, 1\}$, *and* cmd*: (1) if* $\mathsf{EffTamper}(\mathsf{cmd}) = 0$, *the client*

accepts except with negligible probability; (2) if $\mathsf{EffTamper}(\mathsf{cmd}) = 1$, *the client outputs* $\perp$ *except with negligible probability. Both bounds depend only on Merkle binding, the hiding of* tag_q, *the final commitment, and the AuthGC parameter* ρ, *not on* $\mathbf{v}_q$.

Proof. **Case (1).** With $\mathsf{EffTamper}(\mathsf{cmd}) = 0$, the corrupted server forwards prescribed shares at every modeled boundary, and AuthGC binding ensures the prescribed circuit outputs except with probability $2 \cdot 2^{-\rho}$, after which the final commitment check passes.

Case (2). Suppose $\mathsf{EffTamper}(\mathsf{cmd}) = 1$.

PP input boundary. PP recomputes the Merkle leaf and root from the ciphertext bucket, the bucket index in st_Q, and the table-key share. Any non-zero substitution changes the reconstructed root unless Merkle binding breaks.

PP–TK boundary. PP outputs $rt' = rt \oplus H(\mathbf{id} \| \mathbf{freq} \| tag_q)$ with $tag_q = H(\mathbf{b} \| r)$. Since r is fresh and $st_Q = (r, \mathbf{b})$ is XOR-shared, tag_q is hidden from one server, so any change to $(\mathbf{id}, \mathbf{freq})$ or rt' alters the recovered root except with negligible probability.

Final output share. The client recomputes $cm^* = H(\mathbf{id}_{\text{top-}k} \| tag_q \| rt_I)$ from its local $(\mathbf{b}, r)$ and rt_I. Any modified output that is accepted must preserve cm^* or yield a hash forgery, both negligible.

All exceptional probabilities are independent of $\mathbf{v}_q$. □

F.2.4 Proof of Theorem 7.2.

Proof of Theorem 7.2. Fix b, z, admissible $k \in \mathcal{K}$, and any PPT $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. Since z is given unchanged in both worlds, the proof only simulates the transcript and accept bit of the honest challenge query, without $\mathbf{v}_q$.

Let $\mathsf{Sim}_{\mathsf{PIR}}$ be the simulator from single-server DPF-PIR privacy. Define $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$:

$\mathsf{Sim}_0(1^\lambda, I, st_D, k, z)$: invoke $\mathsf{Sim}_{\mathsf{PIR}}$ to obtain $\tilde{q}^{(b)}$ with the correct public lengths; sample $\widetilde{[\![st_Q]\!]^{(b)}}$ uniformly; output and store $\mathsf{st}_{\mathsf{Sim}} \leftarrow (I, st_D, k, \tilde{q}^{(b)}, \widetilde{[\![st_Q]\!]^{(b)}})$.

$\mathsf{Sim}_1(\mathsf{st}_{\mathsf{Sim}}, \mathsf{cmd})$: produce simulated PIR messages and AuthGC transcripts with the public lengths and control flow of the real execution; output $(\tilde{\tau}^{(b)}, 1 - \mathsf{EffTamper}(\mathsf{cmd}))$.

We show $\mathsf{REAL} \overset{c}{\approx} \mathsf{IDEAL}$ via hybrids $\mathsf{H}_0, \ldots, \mathsf{H}_4$.

H_0*: real distribution.*

H_1 — *abort-bit simulation.* Replace β by $\tilde{\beta}(\mathsf{cmd}) := 1 - \mathsf{EffTamper}(\mathsf{cmd})$. By Lemma F.4, $\Pr[\beta \neq \tilde{\beta}] \leq \mathsf{negl}(\lambda)$ uniformly in $\mathbf{v}_q$, and $\tilde{\beta}$ is computable from cmd alone, hence by Sim_1.

H_2 — *query-side simulation.* Replace $[\![st_Q]\!]^{(b)}$ by a uniform string (perfectly hiding by XOR sharing) and $q^{(b)}$ by $\tilde{q}^{(b)} \leftarrow \mathsf{Sim}_{\mathsf{PIR}}$ (computationally hiding by single-server DPF-PIR privacy). Hence $\mathsf{H}_1 \overset{c}{\approx} \mathsf{H}_2$.

H_3 — *AuthGC transcript simulation.* Replace the corrupted server's PP and TK AuthGC transcripts by simulator-generated ones, using the simulated corrupted server inputs and the prescribed corrupted-party outputs. By malicious-secure AuthGC, $\mathsf{H}_2 \overset{c}{\approx} \mathsf{H}_3$; all internal values—plaintext buckets, Merkle paths, candidate lists, frequency counters, and the PP–TK boundary state—remain hidden.

H_4 — IDEAL. The adversary now sees only the public index state, k, z, a simulated PIR view, a uniform query-state share, simulated AuthGC transcripts, and $\tilde{\beta}$. This is exactly $\mathsf{IDEAL}_{\mathcal{A},\mathsf{Sim},D,k,\lambda}(z)$.

Composing, $\mathsf{H}_0 \overset{c}{\approx} \mathsf{H}_1 \overset{c}{\approx} \mathsf{H}_2 \overset{c}{\approx} \mathsf{H}_3 = \mathsf{H}_4$, so $\mathsf{REAL} \overset{c}{\approx} \mathsf{IDEAL}$. □

F.3 Dataset Confidentiality: Formal Treatment and Proofs

This appendix gives the formal real/ideal-style definition underlying §7.3 and the full simulation proof of Theorem 7.3. The collusion adversary corrupts the client and one of the two servers S^*.

Scope. We define dataset confidentiality on a *per-query* basis: the guarantee bounds, for any individual query, what the collusion adversary additionally learns about DB beyond previous queries' outputs and the public state. This matches the threat model of §4.2, which explicitly puts cross-query inference (e.g., adaptive query crafting that combines outputs across multiple queries) outside scope.

F.3.1 Setup.

Per-query view of the collusion adversary. Let DB denote the plaintext database, and let I denote the encrypted searchable index derived from DB, and let q denote a single client query. For a collusion adversary $\mathcal{A}$ corrupting the client and server $S^* \in \{S_0, S_1\}$, define its per-query view as

$$\mathsf{View}_{\mathcal{A}}(\mathrm{DB}, q) := \big(\rho,\ \mathsf{msg},\ \mathsf{tr},\ (\mathbf{id}_{\text{top-}k}, cm)\big),$$

where ρ is the local randomness of the corrupted parties during this query, msg the messages received by them, tr the interaction transcript (including the AuthGC transcript on the corrupted side), and $(\mathbf{id}_{\text{top-}k}, cm)$ is the protocol output the (corrupted) client recovers.

Public authenticated state. Let st_D denote the public authenticated state published by the data owner. In our instantiation, $st_D := rt_I$ is the global Merkle root commitment to the outsourced index I.

Per-query leakage function. Per-query dataset confidentiality is defined relative to

$$\mathcal{L}(\mathrm{DB}, q) := \Big(\underbrace{|I|,\ s,\ l,\ p,\ k,\ st_D}_{\text{static}},\ \underbrace{\mathbf{id}_{\text{top-}k},\ cm}_{\text{per-query output}}\Big),$$

where the static component is fixed by the deployment ($|I|$ encrypted index size, s normalized bucket size, (l, p, k) public protocol parameters, st_D public commitment), and the per-query output $(\mathbf{id}_{\text{top-}k}, cm)$ is the value the honest client would receive, with $cm = H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, st_D)$ and $tag_q = H(\mathbf{b} \,\|\, r)$ derived from the client's per-query randomness r. No plaintext bucket contents, embeddings, intermediate candidate lists, or internal MPC values appear in $\mathcal{L}$.

Quantitative bridge to functional baseline leakage. Among the components of $\mathcal{L}(\mathrm{DB}, q)$, only the per-query output $(\mathbf{id}_{\text{top-}k}, cm)$ carries information about DB that is not already public; the remaining static component depends on DB only through its size and public commitment. Furthermore, cm is a hash of $(\mathbf{id}_{\text{top-}k}, tag_q, st_D)$ with tag_q independent of DB, so the DB-dependent information in cm is fully captured by $\mathbf{id}_{\text{top-}k}$. By Definition 7.1, the non-public DB-dependent information conveyed by $\mathbf{id}_{\text{top-}k}$ to a client correctly executing the (c, k)-ANN functionality is upper-bounded by $\mathcal{L}_{\text{fb}}$ per query. Thus, the leakage function contains no non-public database information beyond the functional baseline leakage.

F.3.2 Formal definition.

Definition F.4 (Dataset Confidentiality, formal). *Protocol* Π *achieves* per-query dataset confidentiality *with respect to leakage* $\mathcal{L}$ *if for every PPT adversary* $\mathcal{A}$ *that corrupts the client and at most one server* $S^* \in \{S_0, S_1\}$*, there exists a PPT simulator* Sim *such that for every* DB *and every single query* q*,*

$$\mathsf{View}_{\mathcal{A}}(\mathrm{DB}, q) \overset{c}{\approx} \mathsf{Sim}\big(\mathcal{L}(\mathrm{DB}, q)\big).$$

The combination of this indistinguishability and the $\mathcal{L}_{\mathrm{fb}}$ *bound on* $\mathcal{L}(\mathrm{DB}, q)$ *established above yields that* $\mathcal{A}$ *learns at most* $\mathcal{L}_{\mathrm{fb}}$ *bits about* DB *per query, beyond the public state. Cross-query inference is out of scope.*

Coverage of client deviations. For the client-deviation aspect, we address the same concern as [43, Claim 4]: a deviating querying client should learn no more database information than an honest client would learn from the corresponding query output. Definition F.4 already quantifies over all PPT adversaries $\mathcal{A}$ corrupting the client and at most one server, and therefore includes adversarially generated client messages, such as malformed DPF keys, inconsistent query-state shares, fabricated st_Q components, and adversarial randomness.

We reduce these deviations to the simulation argument in the proof of Theorem 7.3. Once the adversary's client-side messages are fixed, they are treated as adversarial inputs to the PIR/AuthGC execution. The proof then uses IND-CPA security of bucket encryption (to simulate the encrypted index and the PIR aggregation shares it induces) and malicious security of AuthGC [47] (to simulate the post-processing transcripts) to reconstruct the corrupted-server view from the leaked output $(\mathbf{id}_{\text{top-}k}, cm)$. Moreover, $cm = H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, st_D)$ adds no non-public database-dependent information beyond $\mathbf{id}_{\text{top-}k}$, except with negligible random-oracle collision probability, since tag_q is determined by the client-side query state and st_D is public. Thus, even under arbitrary client deviations, the adversary learns no more DB-dependent information than an honest client receiving the corresponding output, whose leakage is bounded by $\mathcal{L}_{\mathrm{fb}}$ per query.Compared with [43], the only difference is that VeriANN enforces the analogous leakage suppression inside AuthGC, rather than by algebraic masking.

F.3.3 Proof of Theorem 7.3.

Proof of Theorem 7.3. Fix any PPT collusion adversary $\mathcal{A}$ corrupting the client and server S^*, and fix a single query q. We build Sim that on input $\mathcal{L}(\mathrm{DB}, q)$ outputs a simulated $\widehat{\mathsf{View}}_{\mathcal{A}}$ via four hybrid transitions.

Step 1: Encrypted index and derived commitments. From $|I|$, s, and the public index parameters in $\mathcal{L}$, Sim forms a dummy encrypted index $\widehat{I}$ of the same shape and total length by encrypting all-zero buckets of length s under fresh random keys. It samples the corrupted server's key shares uniformly, matching the distribution of one share of an XOR-shared table key. It also builds same-depth Merkle trees over the dummy encrypted buckets, so the stored tree nodes and authentication paths have the same public structure and lengths as in the real index. By IND-CPA security of the bucket encryption, replacing real encrypted buckets with encryptions of zeros yields a computationally indistinguishable distribution; the Merkle nodes are hashes of same-shaped leaves and reveal only public structural parameters.

This hop further propagates to the derived public quantities: let $\widehat{rt}_m$ denote the simulated per-table roots, $\widehat{st}_D = H(\widehat{rt}_1 \,\|\, \cdots \,\|\, \widehat{rt}_l)$ the corresponding outsourced-index commitment, and $\widehat{cm} = H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, \widehat{st}_D)$ the corresponding final commitment, where tag_q is computed from the corrupted client's per-query randomness. Indistinguishability of $(\widehat{st}_D, \widehat{cm})$ from (st_D, cm) follows from IND-CPA propagated through the deterministic hash chain. Sim publishes $\widehat{st}_D$ as the public commitment observed by the corrupted parties; the encrypted index, the Merkle tree, the published commitment, and the final commitment remain mutually consistent within the simulated world.

Step 2: PIR aggregation shares. After Step 1, the corrupted server's PIR aggregation shares $a^{(*)}_{m,n} = \bigoplus_t \mathsf{DPF.Eval}(dk^{(*)}_{m,n}, t) \cdot (T_{m,t} \,\|\, \mathsf{Merkle.ProveInclude}(Tree_m, t))$ are a deterministic function of three components: (i) the DPF keys $\{dk^{(*)}_{m,n}\}$ chosen by the corrupted client, (ii) the bucket-key share $[\![sk_{T,m}]\!]^{(*)}$ provided by the data owner at build time and already simulated in Step 1, and (iii) the encrypted index T_m together with its Merkle tree, both replaced by the simulated $\widehat{T}_m$ and its same-shape tree in Step 1. The simulator therefore produces $a^{(*)}_{m,n}$ by faithfully running PIR.Eval on $(dk^{(*)}_{m,n}, \widehat{T}_m)$ and the simulated key share. Computational indistinguishability from the real aggregation shares follows directly from the IND-CPA hop of Step 1 applied to the bucket contents that PIR.Eval aggregates; no additional DPF-privacy assumption is invoked, since $dk^{(*)}_{m,n}$ is adversary-chosen rather than honest-client-chosen.

Step 3: AuthGC transcripts. All post-processing—bucket decryption, Merkle-root reconstruction, frequency counting, top-k selection, and tag computation—is performed inside the malicious-secure AuthGC instances for PP and TK. The simulator invokes the AuthGC simulator separately for the two instances, using the corrupted server's simulated inputs from Steps 1–2 and the leakage-provided final output $(\mathbf{id}_{\text{top-}k}, cm)$. The intermediate PP output and all internal wires, including plaintext buckets, Merkle paths, candidate lists, and frequency counters, are not revealed in the clear and are covered by the simulated AuthGC transcripts. By the simulation-based malicious security of AuthGC [47, Thm. 5.1], these transcripts are computationally indistinguishable from the real ones.

Step 4: Client-visible outputs. Sim forwards $\mathbf{id}_{\text{top-}k}$ from $\mathcal{L}$ to $\mathcal{A}$, and forwards the simulated commitment $\widehat{cm}$ re-derived in Step 1. The corrupted client locally recomputes $cm^* = H(\mathbf{id}_{\text{top-}k} \,\|\, tag_q \,\|\, \widehat{st}_D)$ using the published $\widehat{st}_D$ and accepts since $cm^* = \widehat{cm}$ by construction. Computational indistinguishability of $(\mathbf{id}_{\text{top-}k}, \widehat{cm})$ from the real $(\mathbf{id}_{\text{top-}k}, cm)$ is inherited from Step 1.

Combining. Each of the four transitions is computationally indistinguishable, hence $\mathsf{View}_{\mathcal{A}}(\mathrm{DB}, q) \overset{c}{\approx} \mathsf{Sim}(\mathcal{L}(\mathrm{DB}, q))$. By the quantitative bridge in §F.3.1, the only DB-dependent information in $\mathcal{L}(\mathrm{DB}, q)$ beyond the public state is upper-bounded by $\mathcal{L}_{\mathrm{fb}}$ (Definition 7.1). Therefore $\mathcal{A}$ learns at most $\mathcal{L}_{\mathrm{fb}}$ bits about DB per query. □

Remark on adversary scope. Definition F.4 explicitly grants $\mathcal{A}$ the client's view, modelling the realistic RAG scenario in which a malicious server registers as a client and combines its server-side

transcript with the protocol outputs the client receives. Without this combined view, the adversary would not be able to even *see* $(\mathbf{id}_{\text{top-}k}, cm)$, and the resulting guarantee would be strictly weaker than what VeriANN actually achieves.